Department of Computer Science

Technical Report

Undergraduate Dissertation: Agent-based simulation of organisational learning

Amy Marshall

Technical Report 2007-11 **June 2007**

ISSN 1740-9497

Contact Address:
Department of Computer Science
University of Bath
Bath, BA2 7AY
United Kingdom
URL: http://www.cs.bath.ac.uk

ISSN 1740-9497

Agent-based simulation of organisational learning

Amy Marshall

Bachelor of Science in Computer Science with Honours
The University of Bath
May 2007

Agent-based simulation of organisational learning

Submitted by: Amy Marshall

COPYRIGHT

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Signed:

Abstract

The aim of this project is to research how organisations move over a fitness landscape over time, looking at the differences between how a singular organisation moves alone and how multiple species of organisations move in conjunction with each other as they co-evolve. This is accomplished by using Kauffman (1993)'s NK and NKC Models and extending them in order to make them more relevant to organisations. The simulation runs carried out first docked the models to Kauffman's, in order to show their validity. Following this, further research is conducted in a number of different areas, investigating some of the extensions made. Within the conclusion a comparison is made between the NK and the NKC model, indicating the importance of both models in furthering this research.

Contents

List of Figures

List of Tables

Acknowledgements

I would firstly like to thank my two supervisors Dr. Julian Padget and Prof. Richard Vidgen for coming up with such an interesting and stimulating project and for guiding me (successfully) through it. It was not at all what I expected, but it has definitely been the most interesting project I have undertaking whilst being at university.

Following on from that I must thank many people for the use of a magnitude of CPU power! Firstly, I must mention my Mother and Father, for lending me all the computers in the house for three weeks, and also all the computers in my Fathers office for the Easter bank holiday weekend. Secondly, I should thank my friends Richard Walklate (for running many simulations), Sam Crawford (for the loan of two servers) and Johnathan Mason (for being Jon). Alis, agentcities and 2E1.14 also deserve a mention here, as (whilst they were in fact two computer science servers and a computer lab) they were also life savers.

I would also like to thank a few people for proof reading sections of my dissertation, primarily Dr. Ruth Marshall (my mum) and also Dr. Alywn Barry and Jenny Marshall. Lastly I would like to Perdita Robinson for her consistent supply of good advice.

Chapter 1

Introduction

"The creation and transfer of knowledge are a basis for competitive advantage in firms." Argote (2000)

Individually, organisations improve themselves and gain advantage over their competitors, through acquiring and then utilising knowledge. However, this is a very simplistic view, looking at this on a larger scale, organisations can be seen as co-evolving with each other and competing firms continually try to improve themselves in order to stay ahead of the opposition. For example; even if an organisation is improving their product, in the long term their net profit will not alter if the competition is matching this improvement. The understanding of this process is important in both the world of management science and of business.

Kauffman (1993)'s NK and NKC Models are commonly used to explore this area. The NK model focuses on an individual organisation and how it improves itself over time, whereas the NKC model focuses on co-evolving organisations, and how they improve or worsen in relation to each other. The limitation of these models is in their biological base and without further extension they can only have limited use. Using past research however, into both the background of these models and into Complex Adaptive Systems, we can justify many extensions.

This paper will explore the NK and the NKC models and their applicability to modelling how organisations interact. Firstly background research must be conducted in order to gain an understanding of the functionality of the models and how they have been adapted for management science in the past. Secondly hypotheses will be made and implementations for the NK and NKC models constructed. The models can then be extended for use in researching organisations, using some of the suggestions made by other researchers. Following this, the models must be verified for correctness by docking the simulation results to Kauffman (1993)'s original conclusions. Finally, the extensions made can be explored and the results gathered used to support or disprove the original hypotheses.

Chapter 2

Literary review

2.1 Introduction

The following literary review should convey an understanding and background to the thesis of this research project; ”agent based simulation of organisational learning”. This understanding being in terms of both management science and of how the research will be conducted with regards to the models and technologies used.

The review is split into three sections. Firstly looking at some necessary definitions within management science and discussing how these allow us to relate organisations to a biological model; such as the NK Model. Secondly a detailed description of both the NK and NKC Models (which are to be used in the research) will be given. During this section possible extensions and adaptations to the models will be researched, in order to make the models more appropriate to organisations. This will be followed by a discussion of how the models have previously been used in research and what affect the past research has on the project. Lastly, agent-based modelling will be discussed and its appropriateness for this research determined.

2.2 Organisational theory

Before we can discuss Organisational Theory we need to have an understanding of what we mean by an organisation. In this research the term ‘organisation’ could symbolise literally any type of firm, company or business. In this way we should be able to represent something as ordinary as a coffee shop or something as unique as a school for ventriloquism, the possibilities are endless. However the type of organisation should (in the model created) not matter, the results should be based on the complexity of the organisation and the method through which it attempts to gain higher fitness, not the type. This should therefore make a set of results applicable to all organisations of the same defined complexity.

As we begin to understand how we might model an organisation acquiring and utilising knowledge, it is important to understand what we mean by the terms organisational knowledge and organisational learning. It is also important to understand the theory behind Complex Adaptive Systems and how organisations fit into this group, as it is this that enables us to apply Kauffman’s model to these situations.

2.2.1 Organisational knowledge

Organisational memory is the term most often used to describe the knowledge an organisation stores. Brooks (1996) describes three types of organisational memory:

- Temporal Memory is information that is accumulated and then transferred from person to person over time. This is knowledge stored within the individuals in an organisation and the passing of information between individuals, more often than not, happens through situated learning. As described by Clancey (1995), situated learning is how individuals learn through being in a situation or carrying out a task. Each individual's learning plays a role in the learning of the whole organisation and as each individual learns, the knowledge of the organisation grows.
- Memory Involving Records is information that is fixed over time in physical records, for example books, papers, codes of conduct and documentation. Absorption of this form of written text is what is more commonly thought of as learning and it is a form of learning that organisations also partake in.
- Structural Memory is based in the structure of an organisation and is a form of organisational memory that adapts over time. Brooks (1996) mentions that Structural Memory "may be considered synonymous with organisational learning" here the organisation as a whole is learning rather than just the individual.

With the models constructed for this project we are seeking to model the fitness of an organisation at a specific point in time and this is a function of both the knowledge of the organisation as a whole (Structural Memory and Memory Involving Records) and the knowledge of the individual (Temporal Memory).

2.2.2 Organisational learning

For an organisation, learning is both the creation of new knowledge and the transfer of knowledge such that the organisation learns from the discovery of individuals. McKelvey & Yuan (2004) define organisations as continuous learning entities; they have to continue learning to stay at the top of their league. The question then is; how this can be accomplished?

Argote (2000) suggests that organisational learning can be facilitated by physically moving knowledge reservoirs (i.e. physically moving employees, tasks and tools around an organisation). McKelvey & Yuan, however, regard organisational learning as already being facilitated by the affects of situated learning (learning from a task or situation), something which their research shows can be improved upon by changing levels of interconnectedness within an organisation (dependant on its size). This level of connectedness is, in fact, something that can be changed by varying the parameters N and K of the NK Model, which McKelvey & Yuan also use to research this (see section 2.3 for a description of both the NK Model and the parameters N and K). McKelvey & Yuans's ideas are likely to be more appealing in business than Argote's, as continuously changing organisational structure, by physically moving employees etc. around, is disruptive and often inappropriate.[1]

It is very easy to claim that as an organisations learns to better itself, and as the individuals learn and pass on their knowledge, the organisation's fitness will improve; this is exactly what is modelled by

[1] There are exceptions to this rule: in consultancy, continuously changing teams and technologies is widely practised, as projects are often short, and require different skills and knowledge. This is not the case in many other businesses, teams are usual set, individuals are employed for a specific role and teams and roles change rarely, as expertise is acquired and horded.

Kauffman's NK Model. Reality, however, is not that simple, the learning processes of other organisations also need to be taken into account. The rate of learning, and the path of learning, of one organisation may easily affect the amount that is gained from the learning of another. It is this environment we will be trying to simulate with Kauffman's NKC Model.

2.2.3 Complexity theory: Complex Adaptive Systems

Next we will reflect upon the meaning of the phrase "Complex Adaptive System" (CAS) and attempt to show that this is what an organisation is. The discussion will then reflect on the impact this has on research into the area.

Tivnan (2005) gives one of the more simplistic definitions of a Complex Adaptive System. He defines a CAS as a system that:

- Has many interacting components
- Equates to more than the sum of its components
- Has the capacity to adapt to its environment

Management science has long believed that organisations cannot be truly understood by breaking them down into their individual parts and seeing how these parts work separately from the whole. McCarthy (2002) suggests that complexity theory is about "understanding the relationship between the whole system and every part of the system as it evolves, learns and adapts" this is because, as Burton & Carroll (2000) state, it is assumed that when looking at the system as merely a sum of its parts, information is lost about how these parts interact.

McKelvey (1999) gives a metaphor of a desk to help describe this, whereby he asks us to think of a firm as we would think of a desk. We know that whilst a desk is a single object, it is also a system made up of many atoms. This idea should then be easily transferable to a firm; a firm is not just a single entity, it a collection of departments, individuals, tasks and process.

A more substantial description of complexity theory, as presented by Anderson (1999), is of individual agents within a system that only have a local view of what is best for themselves. These agents cannot see the whole system, or know what is best for the organisation as a whole. An organisation is able to adapt to its environment because individual agents adapt to improve their own fitness or payoff in each situation they face. It is this idea that is represented by Kauffman (1993) in the NK and NKC Models, where by at each time step, an individual part of the organisation may change to improve itself. The difference is, that whilst there is still only a local view in Kauffman's models, the change is not made if the current organisational fitness is not improved.

Although a number of views on both what a CAS is and how this is applicable to organisations have been presented, they all agree on one thing; an organisation is a Complex Adaptive System, and this is the most important point to draw form this research. As previously mentioned, it is being able to apply the idea of Complex Adaptive Systems to organisations, that allows us to use Kauffman's models to represent them.

Having discussed both what a CAS is and how this is applicable to organisations, it is beneficial to discuss two of the main ways complexity levels can be modified within a CAS. Firstly through changing the number of connections within the system (changing the parameters N and K in the NK and NKC Models). Secondly through modelling a CAS as co-evolving with other CAS (as is the idea behind the NKC Model).

- Burton & Carroll (2000) research the interconnectedness of organisations, investigating the behaviour associated with having a different number of connections. They provide evidence to support their conclusions regarding the level of interconnectedness that is beneficial and also the amount that is needed before the connections become detrimental to the system.
- Co-evolution between organisations can be understood by exploring how organisations are linked together, because of their interaction. This can occur such that as one organisation improves itself to reach higher fitness, another will consequently either increase or decrease in fitness, without physically doing anything to cause this change.

At this juncture we now understand both what complexity is and how to change it. The questions then is; why do this? Complexity can be a dangerous thing for organisations, too much complexity can lead to chaos (further described in section 2.3.6). But firms are looking for their competitive advantage, their success criteria and within this, complexity *is* a necessary evil. McKelvey & Yuan (2004) are convinced the "success criteria" is situated learning, as are Burton & Carroll (2000). Not all authors agree, for example Solow (2003) regard the most important factor in a firms success as its employees and managers. What most research does agree on however, is that the competitive advantage comes not from the success criteria itself, but from the fact that it is something that other organisations do not have and something that is hard for them to copy because of its complexity.

Rivkin (2000) investigates the affects of companies trying to "mimic" / learn from the strategies of other companies. His insight is that it is the complexity of an organisation's business strategy that prevents other companies from completely copying them. Rivkin argues that the "sheer complexity of a strategy can raise a barrier to imitation." The more links between properties of an organisation, the more complex the strategy of the organisation is and the harder it is to "mimic" this. Rivkin likens an organisation to a recipe; whilst eating a dessert, you might be able to work out the ingredients but from there you still have a long way to go before you can recreate it.

We could also apply Lazer & Freidman (2005)'s currently generalised CAS research to organisations, giving a directly opposing slant to Rivkin. Lazer and Freidman research different communications networks and how one entity might copy or "mimic" another through communicating with it. This is not realistic with regards to organisations, as Rivkin says they are too complex to be copied this easily and you can see real world examples that support this, for example companies that far out perform others in the same area. How would something like this be achieved if copying another organisation was easily accomplished? At the same time, the affects of communications networks should not be dismissed, organisations do communicate and may therefore attempt to "mimic" each other, even if the attempt fails to achieve the desired results.

Rivkin's arguments about organisations not being able to mimic each other in this fashion, are actually in line with how the NK Model is designed, meaning that by default it should be his arguments that are followed if the NK Model is used. However that does not stop us adapting the model to incorporate some form of communication in order to investigate Lazer and Freidman's ideas (as in fact they themselves do). Whilst it should not be possible for an organisation to identically copy another (because of the complexity), Lazer & Freidman (2005)'s research can be modified by making it possible to pass only some information (not all). This should give a better reflection of the real world than total silence or total communication.

2.3 Modelling networks

The model presented here is Kauffman's NK[C] Model, originally used in biology to research genes and genotypes, it also has a prominent applicability to management science. It has been used in many studies since it was first introduced by Kauffman (1993) in his book "The Origins of Order". Solow et al. (2002) use Kauffman's NK Model to show "Managerial Insights into the Effects of Interactions on Replacing team Members". Rivkin & Siggelkow (2003) use the same model to investigate centralised decision making within firms. More recently McKelvey & Yuan (2004) use the model to investigate the effects of situated learning in organisations.

The NK Model can easily be applied to organisational learning for two reasons. Firstly, the model very much follows the ideas of the Complex Adaptive System as defined by Tivnan (2005) and, as has already been established, an organisation is a Complex Adaptive System. Secondly, organisations learn in order to improve themselves and the model is designed to represent systems getting fitter over time by adapting traits.

The following discussion regarding Kauffman's models is split into six sections. To give a short brake down of this:

- We will begin with a description of the NK Model, this is discussed in great depth as it is a basis for both the rest of the discussion in this section and the research carried out in this project.
- Following this, a definition of a fitness landscape will be given, going into detail regarding how this landscape is created and also how an organisation within the model might traverse the landscape.
- Next the NKC Model is described, extending the NK Model, it is this model that allows the incorporation of an organisations co-evolutionary partners, its suppliers, consumers and competitors.
- Changing the parameters N and K, something very commonly done by past research, will then be presented and past findings analysed.
- Improvements and adaptations to the system will then be suggested, in order to make the models more appropriate for organisations.
- The last section will then discuss some already discovered behavioural aspects of the models, such that these can be refereed to and recreated by the research conducted, in order to show the validity of the models created in this project.

2.3.1 The NK Model: a detailed description

Below is a description of the parts that make up the NK Model and an example of a coffee shop is included, to further the understanding of the use of this model in management science:

- N: the number of characteristics an organisation has. For example; the different characteristics of the coffee shop might be (1) the employees, (2) the quality of coffee, (3) the cost of running the shop, (4) the selling price of the coffee and (5) the profit the shop makes, in this case N therefore would be five.
- K: how many links each characteristic (N) has with other characteristics, and how they constrain each other. For example; within the coffee shop, the selling price and the cost of running the shop will directly affect the profit made, we could then say K = 2 (but only if every other characteristic was also affected by two others in this way).

- Distribution of K: which K characteristics are used as the links for each N. This can either be the K nearest neighbours of N (wrapping around if necessary) or can be drawn randomly from the set of N. Kauffman shows that the distribution of K is relatively unimportant, as it has little or no affect on the results gained in the different scenarios.
- A: the number of states each characteristic can be in. In the simplest case this is normally two, A = $\{0,1\}$. For example; each characteristic of the coffee shop will have a number of different states that will give different levels of fitness, if this is two states (A = 2) then each employee could be of state "lazy" or sate "hard working".
- f_1: a fitness function to calculate the fitness of a characteristic. The fitness of a characteristic N within the system is dependant on its own fitness and the fitness of K other characteristics of the system. Kauffman reasoned that due to the complexity of this we can model the fitness of a characteristic (and its dependencies) as a random number in [0,1] (this is taken to a fixed number of decimal points). To give a couple of examples:
  - If K = 0 then no characteristic affects any other and each can be individually assigned a fitness. Meaning that whatever state all other characteristics are in, when the coffee quality has a state of "good" this characteristic will always have the same fitness.
  - If K = 1 then each characteristic depends on one other. For example the fitness gained by having good quality coffee may be linked to the price it is sold at, such that the fitness gained by the good quality coffee is different when the selling price is different.
- f_2: a fitness function to calculate the fitness of an organisation. The fitness of an organisation can be calculated by first calculating the fitness of each characteristic in the organisation (using f_1) and then taking an average of these.

2.3.2 A fitness landscape

When run, the NK Model as described above should produce a fitness landscape. To use the model effectively we need to understand the concept behind this and also provide a definition of how a system might evolve and adapt over such a landscape.

The fitness landscape, first introduced by Wright (1932), represents all possible fitnesses of an organisation (when in different states). Fitness, in relation to an organisation, is a measure of how good the organisation is (its performance, profit and cost could all be examples).

To conceptualise this idea of a landscape, let us imagine two extremes, one would be a sand dune in the flat expanse of the desert (highlighting a landscape containing one strong obvious peak), the other would be a mountain range (a landscape full of different sized peaks). As the parameters N and K are increased, the model producs a family of increasingly rugged multi-peaked landscapes. At K = 0, the example of the sand dune in the desert is most fitting, however when K = N - 1 (for some large value of N) the landscape might be better described using a mental image of the Alps.

A landscape is created using the values N (the number of characteristics), A (the number of states possible for each characteristic) and K (the number of connections between characteristics). N and A define the locations available in the landscape (e.g. if N = 3 and A = $\{0,1\}$ the landscape could be defined by the set of locations $\{000, 001, 010, 011, 100, 101, 110, 111\}$). K then defines the fitness of each location (such that each location has a fitness in [0,1]) as described by functions f_1 and f_2 in the NK Model description. Every point on the landscape is a possible state / configuration of an organisation, at a given point in time and every small change an organisation makes to itself, moves the organisation over the landscape.

In order to conduct research using this model a population of organisations (in the case of this paper; 100) are placed randomly on the landscape. Taking it in turns, each organisation can then take an "adaptive walk" over the landscape. There is no algorithm that can ensure that the global fitness maximum is reached by every organisation in polynomial time, as the problem is NP hard. However, as is the nature of organisations, they are not all maximally fit, therefore an approximation algorithm is, in fact, more realistic.

A walk can be taken using one of three methods (Kauffman 1993), but only one move is permitted per organisation for every discrete time step:

- One-mutant change: an organisation chooses a new system configuration from a set of "one-mutant change" neighbours (a neighbouring organisation whose configuration only differs from the current system in one property). If the new configuration can give the organisation a higher fitness, it moves to this configuration, if it cannot then the system stays in its current location.
- Greedy Dynamics: an organisation chooses a new system configuration from the set of "one-mutant change" neighbours, but if the fitness of the new configuration is not fitter than the current location, then the search continues until a fitter variant is found, or all neighbours have been checked.
- Fitter Dynamics: an organisation chooses a new system configuration by looking at every "one-mutant change" neighbour and choosing the one with the highest fitness.

Having walked as far as possible, the organisation will have reached a peak in the landscape, a point at which the current location is in fact higher than all neighbouring locations. From here the organisation cannot move further without taking a long jump. A long jump may be taken by selecting (at random) a completely new system configuration and sending a copy of the organisation to that configuration. The initial organisation will move to the new configuration if it is fitter than its current location. If not, then the organisation will not move and the copy will then continue to jump around the landscape until a fitter variant is found. Each time a jump is made, the time taken to find the next is dramatically longer, because the organisation is getting closer to the optimal peak of the landscape, meaning there are fewer and fewer points in the landscape higher than the current one.

Not all implementations of Kauffman's model in management science have used the idea of long jumps, but Rivkin (2000) does and he asserts that they very much help to model reality. Firms are capable of different types of reorganisation, not only can they adapt by changing one property at a time, they also have the potential to drastically change their strategy. Adding to this argument, organisations do not stop trying just because they hit a set back, because they potentially reach a sub-optimal peak in the landscape. They generally know they are at a sub-optimal peak when they can see their competitors doing better than themselves. Whether or not they do anything about it, will depend on the organisation and the management, but there is definitely the possibility of taking such blind leaps. In fact, this kind of move is quite popular and the consultancy industry has grown to help firms attempt this in a more informed way.

2.3.3 Extending the NK Model: the NKC Model

The fitness landscape of the NK Model is a representation of the landscape one organisation might traverse in its lifetime. However, this cannot be a realistic representation, when there is no influence from other organisations. In reality, an organisation's fitness is affected by the fitness of its suppliers, consumers and competitors. Again, utilising the example of the coffee shop; if Starbucks opens a shop

next door and starts gaining customers, this will mean less business will be going to our coffee shop and its fitness will decrease. Conversely, if there has always been a Costa Coffee down the road and their coffee machine breaks, there will be more possible clients for our coffee shop to serve and its fitness will increase.

The NKC Model, as defined by Kauffman (1993), takes into account the affect different organisations have on each other, the improved model can be defined by introducing:

- S: the number of species of organisation in the system, where by for each species a value for N, K and A must be defined.
- X: the number of other species each species of organisation is connected to. In an initial case Kauffman assumes that each species is totally connected with every other species, however he notes that in reality this would not be so, and that typically each species would interact with a subset of the total number of species.
- C: the number of links between characteristics of different species of organisations. An example of such a link could be: if S_1 is a coffee bean supplier and S_2 is a coffee shop, then the price S_1 sells coffee beans for will effect the price S_2 sells coffee for.
- f_1: a new fitness function to calculate the fitness of a characteristic based on the state of that characteristic and the state of all other characteristics (K and C) linked to it. Because of the affect of the C links, a move of one species across its landscape may affect another species by:
  - increasing / decreasing the fitness of the connected species
  - altering the uphill adaptive walk accessible to the connected species

  We assume that there are no similarities between species, thereby meaning that the effect of S_1 on S_2 can be assigned randomly, in exactly the same way that the affects of K were assigned in the NK Model description.
- f_2: a fitness function to calculate the fitness of an organisation. The fitness of an organisation can be calculated by first calculating the fitness of each characteristic (using the function f_1) and then taking an average of these.

Again a landscape is defined by the parameters N, K and A, therefore there is a different landscape for each species of organisation in the model.

2.3.4 Changing N and K

A vast amount of recent investigations into the model seek to change the N and K values as Kauffman (1993) did and view how this affects the way an organisation can adapt itself / learn in these circumstances. Changing N allows us to look at these affects in organisations of varying size and changing K allows us to experiment with the connectedness within an organisation. Whilst this research is key in industry it has already been very commonly investigated within management science and in relation to organisations; Burton & Carroll (2000), Rivkin (2000) and McKelvey & Yuan (2004) to name but a few examples.

Burton & Carroll (2000) investigate the affects of K, arguing that work needs to be divided as organisations and their environments become more complex and tasks become bigger. It is suggested that this can be accomplished by both dividing tasks and dividing the organisation to deal with the tasks. However,

after this division takes place there needs to be some form of integration to pull the task, the situation and the organisation back together. Burton & Carroll point out the distinct lack of guidance to indicate what level of integration there should be. They suggest that too much connectedness can lead to "over burdened individuals, missed deadlines and higher than expected costs", but at the same time too little can lead to duplicated work and misunderstandings. To avoid both of these undesirable situations the right balance of connections (of K) must be gained. As shown by this example research into different numbers of connections could potentially lead firms to gaining a better understanding of how they should structure their departments and their management.

Rivkin (2000) defines complexity (a high number of interactions) as both necessary and harmful to an organisation. He argues that although tightly coupling different sections of an organisation together makes it harder for other firms to copy their strategy, it also makes it harder to adapt in the face of environmental change. These claims are also a product of his research into changing the K value in the NK Model.

McKelvey & Yuan (2004) focus on the "importance of studying learning as interactions among people in the context of their environment" by specifically researching the amount of learning that takes place and how to increase this rate of learning. They propose that the number of links between characteristics in the NK Model can be directly equated with the number of opportunities one has to learn from others. However, they also state that too many links cause problems due to cost and to organisational components being "bounded rationally".

Within this project a small amount of time will be spent verifying previous research into changing N and K values. The remainder of the research will look at both furthering some of the extensions possible, with regards to the NK Model (mentioned in the next section), and into applying a practical NKC Model, as most previous research, with regards to the NKC Model in management science, is both vague and theoretically based.

2.3.5 Improvements and adaptations to the NK and NKC Models

These are by nature, simplistic models and initially, biological models. As such they were never intended for use in management science and because of this, they leave out many details of importance when looking from this perspective. Different authors have suggested different improvements and extensions to the model, focusing on missed detail they believe is of greater importance. Listed below are some of the extensions considered for this implementation:

- Cost: something mentioned by many authors, is that cost is not taken into account within the model. Within an organisation there is a cost associated with changing to a new system configuration and also with investigating the various system configurations possible. However, in the model defined here, a system can move around the landscape freely, without incurring any cost. Cost could in fact be modelled quite easily, either by limiting the number of steps and the number of jumps available to an organisation, or by introducing a fitness threshold, such that an organisation cannot move unless the fitness gained is greater than this threshold.
- Heterogeneity: another main criticism of the model, this one made by Solow et al. (2005). They suggest that whilst the model assumes the role played by different team members is identical, in reality it is not and that future research should explore situations where team members are heterogeneous. Within the NKC Model, a species of organisation is modelled by one agent, putting great emphasis on the homogeneity of forms within the same species. Kauffman (1993) suggests that "it is possible to extend the model to allow the population representing one species to be a cloud distributed over its landscape".

- A fixed number of states: Solow et al. (1999) suggest that if we are trying to model reality, there needs to be the possibility of more than two states (A) and there also should not be a fixed number of states between different characterises. For example, when modelling different team compositions, in reality there are a variable number of employees available to fill a job role; this might be two, but it just as likely might be one or eight and it will change between job roles. Moving this to organisations, not every characteristic will have the same number of states. For our coffee shop, for example, there might be three states for quality of coffee A = {"excellent","OK","bad"} and only two states for the type of employee A = {"hard working", "lazy"}.

- A fixed number of interconnections: Solow et al. (1999) also argue that in reality there are not a fixed number of interconnections (K) between characteristics of a system. The number of characteristics attached to each N should, in fact, range through different values within a simulation, not just between simulations. They modify the design of the NK Model such that K varies from one N to the next. This idea can just as readily be applied to the NKC Model, by not only applying it to K, but also to C (the number of links between characteristics of different species).

- A weighted contribution of K: Solow et al. (1999) suggest that every K characteristic that affects each N will not affect it to the same degree. For example; if fitness is a measure of how well a product sells, the cost of the product has a different affect on the fitness to the quality. They modify the NK Model to incorporate this by changing the contribution each K has on the final fitness, by giving each a weighting.

- Organisational birth and death: Levinthal (1997) combines organisational adaptation with population selection. He argues, and goes on to show, that the founding form of an organisation within its fitness landscape, has an affect on the future forms possible for that organisation and because of this, a number of dominant forms of organisation emerge. He continues this argument with a claim that the selection process operates differently and that through birth and death it can control a single dominant peak. To facilitate this, he adds two definitions, one to explain the death of an organisation and another to explain its birth. He defines death to occur when an organisation's fitness is beneath a certain threshold, relative to the organisation with the maximum fitness. Birth is then defined as an action to occur in order to replace any death (therefore ensuring the number of organisations on the landscape is constant). Either the new organisation will randomly choose an initial configuration (if average organisational fitness is currently low) or the new organisation will choose to copy a current organisation (if average organisational fitness is currently high).

Researchers still use the NK and NKC Models, despite their many short-comings, as they are robust and simple and have great possibility for the addition of extensions. These models will be used in this project in order to further (and to bring together into one model) some of the modifications both used and suggested by the above authors.

2.3.6 Behaviour of the system

The co-evolutionary process can be defined as the process through which different (but connected) species of organisation change and adapt over time, each affecting the other. This process can lead to one of two states, dependant on values of N, K and C, and the landscape created, either it leads to order or it leads to chaos [2]. In between these two states is the edge of chaos, Kauffman (1993) states that systems that are driven to the edge of chaos, out perform those that are not.

[2]Within the NK Model the system will always lead to order. At some point the highest possible peak will be reached and the organisation will be able to move no further, where as within the NKC Model it may happen that the species will continue moving infinitely.

We need to discover whether or not this theory holds for organisations and we can do this through utilisation of the NKC Model. Before this can be done, however, a better understanding of order, chaos, "the edge of chaos" and the "the edge of catastrophe" must be established.

A system in order: Nash Equilibrium

Mutation in biology is not purposeful, in the way that change within organisations is, yet the patterns of change that take place are very similar. Kauffman (1995) discusses the pattern of change; radical change normally happens early on, as new designs or new strategies are thought of, and then as the process calms down smaller and smaller changes occur. First there is the initial innovation (here Kauffman uses the example of a bicycle). Next there is a phase of constant modification, new ideas and small changes that all give big gains (this is the phase of radical change; the creation of bicycles with big wheels, small wheels, three wheels, one wheel etc.). Eventually, as better and better models are created, small changes and improvements stop having such an affect, there are then a set number of models (e.g. the mountain bike and the racing bike). After this small modifications to these perfected models have little overall affect (for example the most common difference between models of mountain bike now is the colour). At this point there is then order to the system.

Converging toward order produces a stable state, where small changes make no difference to organisational positioning on the landscape. This occurs where Nash Equilibrium is present; each species is better off not changing its current position, as long as no other species changes theirs, which consequently they will not do (Kauffman 1995).

Levitan et al. (2002) describes the ordered regime more formally with regards to organisations and the NK Model. He refers to order as being when each species, S, finds its local or global optimum in the same time step as all other species that affect the payoff of S. Once these mutual optima are attained, further changes to the characteristic states in all species stops, and order prevails. It is this state, that can be thought of as Nash Equilibrium among species of organisation; no species S has a reason to change their own state in response to changes by another species, because no species linked to S is still moving.

This co-evolution towards order, seems at first exceptionally beneficial; if organisations move to peaks in the landscape, they should be able to stay on them. The problem is, that with order, it is also possible for organisations to get stuck at sub optimal peaks, or even below them. Organisations have no control over when they reach an ordered state, or perquisite knowledge about when it will occur, because of the constantly changing nature of the landscape.

A system in chaos: the Red Queen Effect

Converging towards chaos is the exact opposite of converging towards order. Levitan et al. (2002) defines a non-stable or chaotic regime, to be one where adaptive moves by one species, affect the fitness of all other connected species, faster than they can find optimum fitness. This means that the system remains in constant motion, species of organisation chase after continually moving peaks in the landscape and characteristic states continuously change in a failed attempt to increase the species fitness.

When Tivnan (2005) investigated co-evolution and chaos, with respect to organisational learning, he discovered that organisations learn new information and internally store and transfer this information, to enable self improvement and to ensure their survival. Tivnan states that every organisation must do this learning better and faster than its competitors to enable it to stay in the race, this is what is known as the Red Queen Effect. The Red Queen in Carroll (1865)'s "Through the Looking Glass" says to Alice "Now, here, you see, it takes all the running you can do, to keep in the same place", this is what organisations

are doing in the state of chaos. Organisations continuously improve themselves only to stay at the same, constant, rate of fitness. Whilst they themselves are pushing their fitness up, they are also being affected by other organisations' improvement, which is at the same time pulling their fitness down.

The chaotic state is so named, because whilst small changes can send organisations to a peak in the landscape, other (seemingly similar) small changes, can send the same organisation plummeting off the optima, into a valley of the landscape.

Converging towards the edge of chaos

Levitan et al. (2002) claims that achieving optimum fitness over time, seems to rely on finding a balance between the states of order and chaos. This balance can be found at a point near the transition between the stable regime and the non-stable regime. Rivkin & Siggelkow (2003) agree with this, their argument being that to be successful, an organisation must search broadly for a new set of strategies (a suggestion of chaos) but it must also stabilise once it has found a good one (a suggestion of order). To facilitate this a balance is needed between searching for the right strategy (the optimum peak) and stabilising at that optimum, rather than continuing to search on after it has been found.

The reason this balance is necessary is because a system deep in the ordered regime will be too rigid, too frozen into place, to co-evolve away from local peaks. Conversely, as with the Red Queen Effect (a system in chaos), organisations continuously rise and fall over peaks of fitness, never being able to hold themselves upon a high peak. Kauffman (1995) also deems that it is better to keep between the two extremes, at the edge of chaos, as it is at this is the point where peaks are high, but at the same time they can be both obtained and maintained.

The edge of catastrophe

McKelvey (1999) suggests Kauffman is "adding another edge" with his definition of the "Complexity Catastrophe". McKelvey terms this the "edge of catastrophe", whereby going over the edge of chaos stops order and going over the edge of catastrophe stops selection.

As discussed, complexity has its vices, Kauffman (1993) shows that complexity, at too great a level, can undermine a firm's competitive advantage and adaptive capabilities. Order is commonly seen as the consequence of selection, however Kauffman challenges this idea when he defines his "Complexity Catastrophe". He argues that in the following two situations complexity outmanoeuvres selection:

- Situation 1: when an ordered, stable system is produced where the majority of the population is spread over the valleys of the landscape. Here selection cannot keep the population on the high peaks: the system is ordered "not because of selection but despite it" Kauffman (1993). This can occur in the NKC Model, but not the NK Model, as an organisation can only become trapped, beneath a peak of the landscape, through the influence of another.
- Situation 2: when (because K is high and the landscape rugged) the fittest members of the population are not very different from the rest, even with strong selection, as they are trapped on sub-optimal peaks. This can occur in both the NK and the NKC Model when K is high in comparison to N.

Kauffman (1993) claims that increasing interaction (high K) leads not only to decreasing fitness but also to the complexity catastrophe. K is best at a low number and whilst N increases, K also increases (to gain maximum fitness) but much more slowly (the best K is always very small, but greater than zero).

The edge of catastrophe is the value of K at which fitness is maximal, such that fitness will decrease if K either increases or decreases.

Solow et al. (1999) suggests that this occurs because at first an increase in K creates more opportunity for different choices of fitness, however as K increases further, the increasing conflict between the interconnecting parts of the system out weighs this. Solow et al. (2002) supports this, showing that whilst small amounts of interaction are beneficial, large amounts make the situation very unsupportive of change in the environment. This is not a productive quality; change happens constantly within the environment organisations are situated in.

2.4 Agent-based modelling

"An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives" Wooldridge (2005).

As has been discussed in previous sections, the hope is to create two models of how organisations learn and adapt over time, based on Kauffman's NK and NKC Models. In order to then run simulations on these models, organisations will need to be set loose to move around the created landscapes. This idea maps very well onto an agent-based model; the fitness landscape (defined by Kauffman's model) would be the environment that the agents are situated in and the agents themselves would be the organisations. Organisations need to be able to make their own decisions about where to move next on the landscape, and as such it makes sense to make them agents, as modelling them in this way allows them these capabilities.

The rest of this section should give a better definition of exactly what an agent and an environment are, discussing in detail how Kauffman's models can be applied to these. Following this, the desirability of an agent-based modelling toolkit will be addressed.

2.4.1 The environment

Wooldridge's definition of an agent being situated in an environment is very suitable for our purposes. Russell & Norvig (1995) define four different properties an environment might have. Defining the landscape of the NK and NKC Models using these properties, allows us to see the landscape as an environment for the agents:

- Accessible vs. inaccessible: an accessible environment is one in which an agent can get complete, accurate and up-to-date information at all times. In both models the landscape is inaccessible, as the organisation can only query and find information from the one-mutant neighbours of the location it is currently occupying, the rest of the landscape is unknown.
- Static vs. dynamic: a dynamic environment changes throughout the simulation, whilst a static environment stays constant. In the NK Model the landscape is static and does not change once it has been created. In the NKC Model, however, the landscape is dynamic. The environment is different for every different agent and it changes over time depending on the locations of the other agents this one is linked to.
- Discrete vs. continuous: a discrete environment is one where there is only a finite number of states, whilst a continuous environment contains an infinite number of states. In both models the landscape is discrete because there are a finite number of states (even though, for the NKC Model particularly, this is an extremely large number).

- Deterministic vs. non-deterministic: a deterministic environment is one in which every action has a single defined reaction and there is no uncertainty about the state that will occur when the action is performed, a non-deterministic environment is the opposite. In the NK Model the landscape is deterministic, because every action has a single defined reaction. In the NKC Model, however, the landscape is non-deterministic. The movement of each agent will affect the other agents it is linked to, and that agent has no prior knowledge of what affect this will have.

2.4.2 The agent

Wooldridge & Jennings (1995) give eight different properties an agent can have. Thinking of organisations in terms of these properties, enables us to better understand how they can be modelled as agents. The properties of agents include:

- Proactive: a proactive agent is one that acts in order to try and reach defined goals. The agents in both models are proactive, working towards the goal of gaining the highest possible fitness.
- Reactive: a reactive agent is one that reacts to changes in the environment. The agents in the NK Model cannot be reactive, as the environment is static. The agents in the NKC Model however, are reactive, because, as the locations in the environment change in fitness, the fitness function compensates for this.
- Sociable: a sociable agent is one that communicates with other agents in the environment. Naturally the agents in neither model are sociable, but if incorporating the communication networks of Lazer & Freidman (2005), or something similar, they will become more so.
- Mobility: an agent that has mobility is able to move around an electronic network. This is not applicable to this simulation.
- Veracity: veracity is whether or not an agent will knowingly communicate false information (this will never happen in either model of this simulation).
- Benevolence: an agent is self interested if it has conflicting goals and it is benevolent it will help other agents. In the NK Model model, this is not clear cut, as whilst the agents all have the same goal, to move to a higher fitness, but they do it independently. Within the NKC Model, however, species of organisations are very much self interested, any movement they make on the landscape to better themselves, could very easily badly effect another species.
- Rationality: a rational agent will act in order to achieve its goals. In both models agents are rational and will only ever change location if that change is beneficial to them.
- Learning/adaptation: an agent has the ability to learn or adapt if its performance improves over time. Agents in these models do not learn and do not modify the way they work, they follow the same algorithm throughout the simulation.

2.4.3 Agent-based toolkits

It is important, when modelling, to have a level of abstraction. The modeller should not be tied down to coding all the individual complexities of the system, when the research to be done does not require it. The modeller should also not have to develop everything from scratch, code reuse should be possible

as this type of research is common and a lot of the initial building blocks used to create agents and their environments are similar or the same between models.

Whilst using a agent-based modelling toolkit requires time to learn, it saves time in the long run. If the correct tool kit is chosen it can provide multiple advantageous tools, making it possible to define the model at a higher level, meaning that the "low-level details are often completely transparent" Parker (2001). Some examples of possible useful features include: graphical run time support (good for debugging and testing); the facilities to capture and display data; the facilities to investigate or observe agents whilst the models are running; tools and libraries of relevance to different situations (enabling code re-use); facilities to do bulk runs.

Due to the above mentioned advantages an agent-based toolkit will be used to enable faster development of the experimental model. The choice of toolkit will be discussed further in chapter 3.

2.5 Summary

In summary this project will consist of the creation of two agent-based models; the NK Model and the NKC Model. We are able to apply these models to organisations because although the models were originally created for biological research, they are in fact based on Complex Adaptive Systems, which, as determined, is what organisations are.

Extensions will be made to these models in order to make them more applicable to organisations. Extensions suggested will include ones in some of the past research discussed and also ones devised by the author (to be discussed later).

The past research incorporated here will also allow the verification of the models created, by comparing results gathered to past results.

Chapter 3

Selecting an agent based toolkit

The following chapter addresses the choice of agent-based toolkit. Within this, both the options available, and the criteria established for selection are addressed. A choice of toolkit is then made and justified.

3.1 Researching the options

The class of agent based toolkits is a large one; however, as they were not all created with the same purpose in mind, it is not sufficient to randomly choose one. Whilst the critique formed here is not all encompassing, the most relevant toolkits have been mentioned. This initial analysis, enhanced by some of the general toolkit listings and analyses already available, such as Detlor & Serenko (2002) and Hofmann & Tobias (2004), made it possible to discard less relevant toolkits and focus on developing a better knowledge of the rest. The analysis included investigation into the following subject areas:

- Domain: the desired toolkit needed to be either for general use, the social sciences domain or the organisational domain, in order to be relevant for this research.
- User base / Support: the toolkit needed to be adequately supported, including a range of the following: technical documentation, on-line tutorials, reporting facilities and opportunities to ask for help from experienced users. This was in order to ensure that the development of the project was not hindered by lack of information.
- Availability: the desired toolkit needed to be available and free to download.
- Language (Implementation Language / Coding Language): the toolkit needed to be written in a language that was portable and useable on many operating systems and programmable in a language that was either known by the developer or easy to learn.

From here it was possible to narrow the number of toolkits for continued analysis. Table A.1 (in appendix A) gives a short description of the findings of this phase of the research.

3.2 Narrowing the search

Using the information presented in table A.1 and a process of elimination which removed any toolkits that were unsuitable, six toolkits were chosen as fitting the basic criteria for this project. Toolkits were eliminated based upon: inappropriate domains such as Education (Agentsheets and StartLogo) or robotics (SIM_AGENT); incomplete user documentation (VSEit) or too few users (JAS); the toolkit not being readily available (VSEit) or only being available to purchase (AgentSheets); inappropriate implementation and coding languages such as Smalltalk (SMDL, CORMAS), PoPI (SIM_AGENT) or Visual AgenTalk (AgentSheets). Reasoning behind the elimination of any toolkit can be deduced using the information in table A.1.

The six toolkits chosen for continued analysis were Repast, Ascape, Swarm, NetLogo, MadKit and Mason. These were then evaluated in a more detailed manner, based on the following three criteria:

- User base / Support: this was examined in more detail to enable a better comparison between toolkits. Research into what support was available in the form of technical documentation, tutorials, email support, forums, FAQ, downloads, etc. was made.
- The facilities available for collecting and recording data: the eventual aim of this project is to formulate a results set to support theories and hypothesis. To accomplish this it was established that vast amounts of data would need to be collected, therefore the toolkit would need to facilitate this.
- The runtime support for examining the simulation and the runtime graphical user interface: the desired toolkit would firstly need to display results in readable forms such as graphs and charts and secondly need to assist in viewing the workings of model during a simulation run.

Table A.2 (in appendix A) summarises the results from this analysis. Using these results, it was decided that Repast would be the most appropriate toolkit for this project.

3.3 Repast: a final selection

"The **RE**cursive **P**orous **A**gent **S**imulation **T**oolkit (Repast) is the leading open source large-scale agent-based modelling and simulation library" Macal & North (2005)

Of the toolkits investigated Mason and Madkit are both relatively new and untested, Ascape is not well supported, Swarm does not have a runtime Graphical User Interface and Swarm and NetLogo would require extra work in learning a new programming language. There are many reasons not to use the toolkits set aside but Repast was also chosen for the extra facilities it has to offer. Repast fulfils the initial requirements of having a good user base, adequate facilities for collecting and recording data and a runtime Graphical User Interface support. It is also easy to use, has a short learning curve, and is extendible, robust and has good support for network simulation (including default node and edge classes) Collier (2000).

Chapter 4

The NK Model: experimental hypotheses and design

This section will cover the two main tasks that were carried out in relation to the NK Model when designing the experiments that would take place. Firstly the experimental hypotheses will be discussed in detail, looking at why these hypotheses were made and what stimulated their suggestion. Secondly, the implementation of the basic model will be discussed in detail and the implementation of some of trickier extensions to the model will then be broken down.

4.1 Experimental hypotheses

This section will be laid out in three subsections. Before starting to make hypotheses regarding the model, a short description of the model will be given as a memory aid (for a more detailed description, please refer to section 2.3.1 of the literary review). Secondly hypotheses will be made regarding the basic model (the model without any extensions). These aim to show the model as functionally equivalent to Kauffman's Model. Following this, hypotheses will be made that will stimulate further research and require extensions to the basic model.

4.1.1 The NK Model: a reminder

The NK model contains two parts; a landscape (specified by the parameters N, K and A and the functions f_1 and f_2) and a set of organisations (or agents) that walk across this landscape.

- N is the number of characteristics in an organisation.
- K is the number of links that each characteristic has with other characteristics.
- A is the number of states each characteristic can be in.
- f_1 is the function that calculates the fitness of a characteristic in an organisation.
- f_2 is the function that calculates the fitness of an organisation.

- A location in the landscape is a configuration of characteristics, such that every characteristic is assigned a state.

4.1.2 Hypotheses using the basic model

The basic NK Model is the model exactly as it is described above, with no extension. The research that can be done with this alone is mainly a repeat of research that has been done before, which is required in order to show that the model designed and implemented here functions as expected.

Hypotheses one to six (below) attempt to align the model with past research by making statements that are already well supported. Hypothesis seven, however, is intended to initiate an investigation into the affect of changing the number of states, A, that each characteristic can have. Kauffman limits this to two in his model, but removing this limit gives a more realistic simulation of an organisation; the number of states an organisation's characteristics can have, is by no means limited in this way.

Hypothesis 1: as N increases and K stays the same it will take longer to walk to the fitness optimum because there are a greater number of organisational locations to pass through. Whilst it is obvious why if there were more locations, these would take longer to walk through, it is not necessarily as obvious why there are more locations. The best way to show this is with two examples, using different sizes of N; if N = 2 and A = 2 the possible locations include $\{00, 01, 10, 11\}$ however if N = 3 and A = 2 the possible locations include $\{000, 001, 010, 011, 100, 101, 110, 111\}$.

Hypothesis 2: as N increases and K stays the same, the average fitness of the population will also stay the same. Kauffman theorises that this is the case because the fitness values are drawn randomly from between 0.0 and 1.0. Order statistics show that because of this random selection, the average value of the least fit location should be 1/3 and of the most fit location should be 2/3.

Hypothesis 3: When K = 0, at the end of the simulation all organisations will have the same fitness. When $K > 0$, at the end of the simulation there will be different organisational fitnesses. This is because when K = 0 there is only ever one fitness peak in the landscape however when $K > 0$ the landscape is rugged and where there is more than one fitness peak there is the possibility of organisations ending the simulation on different peaks.

Hypothesis 4: as N stays the same and K increases, fitness will be maximal at a low value of K (but not at K = 0). An organisation is better with a small number of interconnections, a large number enforces too much complexity and too much reliance on other elements of the organisation, but conversely an organisation with no interconnections will not function maximally either.

When K = N - 1 the fitness of the population will be sub-optimal for two reasons, firstly the landscape will be very rugged meaning organisations are more likely to get stuck on a sub optimal peak, secondly peaks will tend to have lower fitness because the averaging will over all characteristics will drag the fitness towards 0.5. When K is zero, however, the opposite occurs and there will be only one peak, this means there is a risk regarding the landscape as whilst the peak might be very high, there is just as much of a possibility that it will be very low. With a low value of K (that is not zero) there will be more than one peak, meaning there is more of a chance there will be a high point on the landscape. As each location is not averaged over as many characteristics, this will be a peak further from 0.5 and also as there are not as many peaks, there is a greater chance of more locations reaching one of the higher peaks.

Hypothesis 5: as N increases the value of K that gives highest fitness also increases but NOT at the same rate as N increases. Whilst the value of K will always be a low value in comparison to N (as described above), the K value that gives the higher fitness will however increase (even if very slowly) because larger organisations need more interactions to keep information flowing round them optimally.

Hypothesis 6: fitness of N decreases towards 0.5 as K increases because of the complexity catastrophe (as described by Kauffman and as detailed in the literary review, section 2.3.5).

Hypothesis 7: as A (the number of states) increases, there will be a larger landscape with more locations (because of the increased number of states each characteristic can have) it will therefore take a longer time to reach a fitness peak. Also, the landscape will be more rugged, with more locations, (but unlike K, this will not affect the fitness of the peaks available) therefore the overall fitness of the landscape will stay the same but there will be fewer organisations on the highest fitness peak at the end of the simulation.

4.1.3 Hypotheses requiring extensions to the model

The following hypotheses move the project out of the over explored territory, of merely changing N and K, and into an area where current research is more limited. There are many different extensions proposed and already explored by different authors, but it is not possible with the current time constraints to research all areas highlighted in the literature review. This section should show both what is to be researched during this project and also why this is of particular interest.

Next neighbour method

When walking across the landscape it is necessary to specify exactly how to choose which neighbour to look at next, once the set of all neighbours has been constructed. Reading through Kauffman's specification for the model it is unclear how he does this, though it is very likely he chooses randomly. There are, however, three prominent methods for how to choose the next neighbour, these are:

- Taking the next neighbour from the set of neighbouring locations in a set order each time.
- Taking the next neighbour from the set of neighbouring locations randomly, with no memory of the last one chosen.
- Taking the next neighbour from the set of neighbouring locations randomly, but with a memory of the ones already visited.

In actual fact all of these methods symbolise a way an organisation might attempt to learn and improve itself over a fitness landscape, they are just different methods of doing this. Taking the neighbouring location in a set order each time may seem to go against the randomised methodology of the rest of this model, however organisations do do things in a very structured way and this is one part of Kauffman's work that could model this realistically. On the other hand, modelling this randomly allows for the many different possible options an organisation has at this point and facilitates different organisations acting differently.

Lastly, it should be briefly mentioned why the random method offers the choice of with or without memory. In the NK Model, the fitness of a neighbouring location will not change, so after it has been looked at once, there is no reason to look at it again. However, in the NKC Model, this is not the case; the fitness of locations will change and it may therefore be interesting to compare the results of using this feature within both models.

Hypothesis 8: the method used to find the next neighbour will not have a significant affect on the trends in the results gathered. The two forms of random selection will give the same results; however, if the organisation has no memory of the last location accessed, the process will take longer. Comparing random selection to an ordered selection; in an ordered selection locations are still given a random fitness

and therefore any order that they are selected in will, in essence, be "random". The main difference, when using an ordered selection, is that all organisations that come to the same location will travel the same path, this however affects the population less on the larger landscapes (which organisations typically have), as the landscape is so large that is becomes less and less likely that two organisation's paths will in fact cross.

Realism of K

With regards to modelling K in a more realistic manner there are two elements that can be changed, firstly which K characteristics are used and secondly how many characteristics are used for K.

Kauffman (1993) researches into whether or not which characteristics used make a difference. In his original model the K characteristics used are neighbours of N, however it was argued that this created a bias, therefore Kauffman researched this. Choosing K randomly from the set of N and comparing the results to his originals, he found supporting evidence to his hypotheses that which K used actually made no difference. It is important to verify this research here because, realistically, an organisation would have very specific links between characteristics and it must be established that which K are used will not affect the results of this model, in order for this research to be valid.

Hypothesis 9: it makes no difference whether the K characteristics that affect N are random or are neighbours of N (as show by Kauffman).

In the basic model, the number of links each characteristic can have with any other characteristic is the same, but, as Solow et al. (1999) argues, within an organisation this should not be the case. For example; in a supermarket you could have two characteristics that are affected by a different number of other characteristics:

- For the characteristic N = "employee effectiveness", K might be three: "employee pay", "employee attitude" and "quality of management".
- For the characteristics N = "customer satisfaction", K might be four: "employee effectiveness", "cost of shopping", "availability of desired items" and "quality of items bought".

It is important that the affects of variations in K be researched as they are very relevant to organisations (as can be seen by the example above). Solow et al. (1999) suggests choosing K randomly, using the K that is inputted into the model as the average K across all characteristics. The hypotheses below include looking at this using both a uniform distribution and a normal (Gaussian) distribution of random numbers.

Hypothesis 10: when the number of dependencies K is a random number, with an average of K, the model results will be the same as when using K identically, because the average over a location will be the same. Some characteristics will be very interdependent and some not very, but this will average out to create the same amount of interdependencies as there were when K was identical.

Hypothesis 11: when the number of dependencies K is a Gaussian number mapped to the space plus or minus x (such that $(K + x) < (N - 1)$ and $(K - x) \geq 0$) the model results will again be the same as if K was identical, for the same reasons as mentioned above.

Realism of A

In the basic model each characteristic has the same number of states, in reality this is not the case within an organisation. As Solow et al. (1999) suggests for K, here we will do the same for A. Again, the

differences found when choosing A from a uniform distribution of random numbers and then from a Gaussian distribution of random numbers will be investigated. This will be done using the following hypotheses:

Hypothesis 12: when A is chosen randomly with an average at A, rather than uniformly, this should not affect the model results. This is because changing A will only change the number of locations, NOT their relative fitness and in this instance, there will be a similar number of locations as in the original model, as the average number of states is still A.

Hypothesis 13: when the number of states A is a Gaussian number, mapped to the space $0 - > 2$A, the model results will also not be affected, because again, the number of locations will not change dramatically as the average number of locations is still A.

Calculating fitness

Fitness is usually calculated using function f_2, this calculates the fitness of each characteristic (using the function f_1) and then takes an average over these characteristics. This, however, is not the only method that has been considered, two other version of the function f_2 include: one that takes the fitness of the weakest characteristic and another that takes a weighted average over all characteristics.

It is often said that a group is only as strong as its weakest member, in this instance meaning that each characteristics could be held back by the weakest characteristic that affects it.

Hypothesis 14: when using the weakest fitness rather than an average over all fitnesses the overall fitness will be lower, this is obvious. The real question here is whether or not this gives a more realistic view of how organisations move over a landscape, unfortunately this is not a question that can be proved or disproved by running this model.

Moving on from this, taking a weighted average, as suggested by Solow et al. (1999), could also be applicable to organisations, because it is true that not every characteristic affects another to the same degree.

Hypothesis 15: when taking a weighted average, rather than a uniform average, this *will* affect the fitness of the landscape. As it is random which characteristics are weighted, and by how much, there will be some locations that come out with higher than normal fitness and some with lower, meaning there will be a greater range of different finesses values over the landscape. Higher than normal fitness will come about from characteristics with a high fitness being given a heavy weighting, whilst lower than normal fitness will come about from characteristics with low fitness being given a heavy weighting. In the simulation overall, there will then be a larger than normal maximum fitness and a lower minimum fitness, but the average fitness will still be the same.

Whilst not to be researched here, a last thought should be given to modelling realistically. Having said that both of the above options could be considered more true of an organisation than merely taking an average over the characteristics, it should then be considered in what situations (or for what types of organisation) the two different methods might used.

Walking across the landscape

In the basic model the walk over the landscape is done using the one-mutant change approach, Kauffman (1993). Kauffman does, however, give two other methods of moving over the landscape; greedy dynamics and fitter dynamics.

- One-mutant change: a neighbouring location is selected and if it is fitter than the current location it is moved to.
- Greedy dynamics: the set of neighbouring locations is constructed and the first one found with a higher fitness than the current location is selected and moved to.
- Fitter dynamics: the set of neighbouring locations is constructed, all locations are considered and the one with highest fitness is moved to.

These are three very different types of walk and it is realistic to assume that different organisations might proceed in their walk across the landscape in different ways. The research conducted into these different walk types hopes to gain insight into the final locations found and the speed at which they are found by the different methods.

Hypothesis 16: when using the greedy dynamics approach the final fitness location will be the same as when using the one-mutant change approach, however a stable state will be reached in less steps of the simulation. This is because the same process is taking place, but what happens over many ticks of the one-mutant neighbour method happens over one tick of the greedy dynamics method.

Hypothesis 17: when using fitter dynamics the over all fitness of an organisation on the landscape will tend to be lower because there are alleyways that are always left unexplored i.e. areas of the landscape with a gradual slope that eventually reach a higher peak.

It is not assumed that one of these methods models an organisation better than another, but that different organisations, under different management, might approach the problem differently.

Jumping across the landscape

Allowing organisations to jump over the landscape is an extension Kauffman himself made to the basic model and something many organisational scientists have used, for reasons previously discussed. An organisation will not necessarily stop moving when they reach a peak in the landscape, especially if they can see other firms elsewhere outperforming them. They then know that they are only at a local peak, and because of this it is likely they will want to continue improving themselves.

Hypothesis 18: allowing organisations to jump over the landscape will increase the fitness of the population such that, given enough time, the entire population will reach the highest fitness peak.

Introducing cost and setting limits

It was said that adding jumping into the landscape should give greater realism, but it is not realistic to think that every organisation will eventually reach the maximum fitness peak, as they would if every organisation were allowed to jump infinitely, but this does not happen in the real world. Maybe it is possible, but most, if not all, firms do not have the resources to continue to search in this way forever. Therefore there need to be some limitations on this.

An initial limitation can be set by introducing a fitness threshold, such that an organisation cannot move (jump or walk) to a new location unless that location is greater than the current one by more than the fitness threshold. This will obviously lower the fitness of the population of organisations as a whole, but it is more realistic to assume that an organisation is not going to change its organisational strategy over a very low fitness gain.

Hypothesis 19: increasing the fitness threshold will decrease the overall fitness of organisations on the landscape, not just by the small amount that the threshold represents, but by a greater amount by blocking off potential pathways that exist by going through that location.

More limitations can be set with regards to jumping across the landscape, a limit can be set on the number of successful jumps allowed, and also on the length of time allowed for searching for a new location.

In the business world an organisation is not going to continue changing its configuration forever, as this will incur a unmanageable cost. Therefore, a limit on the number of jumps permissible is likely. It is also unlikely an infinite amount of time will be allocated to search for a better configuration. Time spent on this will cost, therefore there will likely be a limit, such that if the given amount of time is spent searching and no preferable configuration is found then the search will end.

Hypothesis 20: decreasing the number of successful jumps allowed will decrease the fitness of the over all population, again this is something that is obvious but the research into this should show that there is scope for modelling cost within organisations.

Hypothesis 21: increasing the number of characteristics (the size of the landscape) whilst keeping the number of successful jumps the same, will be more limiting and will give lower fitness for larger N. This is because as the search space is widened, it is less likely to find the highest fitness peak in the same amount of time, or in the same number of jumps. However, this phenomena is not necessarily relevant, as larger organisations generally have relatively sized funds, meaning (with respect to the model) that the number of successful jumps possible would likely increase with the size of the organisation (and a higher N is generally associated with a larger organisations).

Hypothesis 22: as the amount of time an organisation can search for increases, the fitness of the overall population also increases. Given a longer period of time to search, an organisation is more likely to be able to find a solution. This increase will reach a limit when the fittest location in the landscape is found.

Hypothesis 23: increasing the number of characteristics (increasing the size of the landscape) whilst keeping the time an organisation can search for the same will be more limiting (give lower fitness) for larger N. This is because increasing N increases the size of the problem space, meaning a larger amount of space needs to be searched in the same period of time.

Life and death of organisations

Levinthal (1997) suggests that both birth and death are very relevant to organisations. Over time some organisations die (they may go bankrupt, get taken over, or just discontinue) and at the same time other organisations are brought to life.

Levinthal models organisational death by using a death threshold. This threshold can be subtracted from the fitness of the fittest organisation within the population. The resulting fitness value can then be used, on a per organisation basis, to decide whether the organisation lives or dies on this time step. If the organisation's fitness is greater than the calculated value the organisation lives, however if its fitness is lower than this value it dies. The amount of death within a population will then depend on the death threshold that is set, in the business world the threshold may depend on both the competitiveness of the industry in question, and on how much the consumers in the industry are willing to compromise between organisations. However, whatever the reasoning behind the differences in this threshold, it is interesting to hypothesise about what affect changing it should have on the population.

Hypothesis 24: the smaller the death threshold (therefore the more death there is within the population) the longer it will take for organisations on the landscape to reach order. This is because new organisations are continuously being born and many of these will immediately die if they begin too low on the

landscape.

Hypothesis 25: birth and death is good for the population, in that it will give a higher fitness to the overall population if the correct death threshold is found. This correct death threshold will neither be very large or very small. A very small threshold will mean lots of death and organisations dying before they have a chance to explore the territories they have been placed in, however a very large (or non-existent) threshold will mean that organisations do not die, no new organisations are born and therefore new territories are left unexplored.

For every organisation that dies, a new one is born to keep the population stable. This is the other interesting aspect in Levinthal's research; the birth of a new organisation, specifically looking into the choice of which starting location the organisation chooses. Three possible methods for deciding this start location will be investigated:

- Randomly choosing a starting location from the landscape.
- Choosing a starting location identical to the location of one of the current organisations (which current organisation to use for this is chosen at random).
- Using both of the above, as Levinthal did; choosing to either copy a current organisation, or to start at a random location, based on the genetic load of the population (the ratio between the maximum fitness and the average fitness). "The probability of a random birth or of a birth via replication is assumed to equal the genetic load of the population." Levinthal (1997). The genetic load is converted into a percentage and then used as the chance that an old organisation will be copied.

Hypothesis 26: when new organisations always copy current organisations the maximum fitness peak is less likely to be found, as new territories are left unexplored.

Hypothesis 27: when new organisations are randomly assigned locations, order takes longer to find than when they copy current locations. However, a higher over all fitness will be found because more new territory is covered.

Hypothesis 28: using the genetic load to determine when to copy a current location and when to create a random new location will find order faster but will still explore new territories when the population is at a generally low fitness.

Realistically organisations might follow any of the given methods above however this research is hoping to show that Levinthal was looking at the most effective approach.

4.2 Implementation of the NK Model

This implementation section provides a detailed description of the basic model on which the foundations of this research is based. A short explanation is then presented regarding the additional functionality for each set of hypotheses. The hypotheses are laid out in the same order and under the same section headings as in the previous section. (For a detailed specification of the NKC model see Appendix B).

4.2.1 The basic model

The basic implementation of the model takes in five parameters before the simulation begins (additional parameters will be mentioned later in the document and a full detail of the use of all parameters can be found in Appendix D):

- N_size_of: the number of characteristics of each organisation (and therefore of each location on the landscape).
- K_size_of: the number of characteristics each characteristic depends on for its fitness calculation
- A_size_of: the number of states each characteristic can be in
- organisations_no_of: the number of organisations that are thrown onto the landscape at the beginning of the simulation (default is 100)
- fitness_range_dp: the number of decimal places the fitness range covers (default is 2 decimal places)

In the literature the landscape is created first, the organisations are thrown onto the landscape and the simulation begins. In reality this is not possible with available resources, as a landscape has A^N locations (configurations of characteristics and states). With numbers as small as A = 2 and N = 30 there are 1,073,741,824 locations, an unfeasible number to create. The way the landscape is created, therefore, is slightly different here to how it is described in the literature. However, the technique described will still provide the same results.

The basic implementation is made up of three classes:

- The class NKModel extends the Repast class SimModelImpl and implements the class SimModel. This class is where the model is set up and is also where each step of the simulation runs from.
- The class NKFitnessLandscape contains all methods to calculate and access the fitness of all locations in the landscape.
- The class NKOrganisation is the agent class, this stores all methods relating to moving the organisation over the landscape.

For the simulation to be run in Repast the program requires one class to be used as the model class. In this case, the NKModel class. All parameters used to set up the model (including the five mentioned above) and the corresponding getter and setter methods for these, are included in the NKModel class. Repast can then use these getter and setter methods to either create a GUI, to allow a user to change parameters on individual runs, OR to interpret a parameter file for a batch run. The NKModel class also provides a build method that is run by Repast before the simulation begins. This sets up a new NKFitnessLandscape, creates all the NKOrganisations to be thrown onto that landscape and sets up the data collection facilities (either real time graphs as the model runs, for use with the GUI, or data exports to .txt files, for use with batch runs). Lastly the model class provides pre-step, step and post-step methods that run on every tick of the simulation, allowing a set actions to occur for each organisation (each agent) on each tick, in this instance each NKOrganisation takes an adaptive walk across the landscape.

The class NKFitnessLandscape stores the landscape the organisations move over during the simulation, in the form of a set of locations and associated fitnesses. Rather than storing the whole landscape for the entire simulation the fitness of each location is calculated when it is needed. This is done using the method getFitness(String key). This method, when given a string of characteristics (the parameter key), takes each characteristic individually and finds its fitness. The method then returns the mean of all characteristic fitnesses.

The method used to find the fitness of each characteristic is characteristicFitness(int N, int K[ ]), where N is the index of the characteristic we are finding the fitness of and K[ ] is an integer array containing the state of that characteristic and of all K characteristics that affect it.

The fitness of a characteristic changes depending upon the state that characteristic is in and the states the K characteristics it depends upon are in. All possible fitnesses of each characteristic are stored in individual hash tables (one for each characteristic).

To find the fitness of a characteristic the hash table for that characteristic must be searched. The unique key into the hash table is a string constructed from the state of N and the states of the K characteristics that affect N. Using this key, the hash table can then be searched. If a fitness relating to the key is found, it is returned. If a fitness is not found, a new random number is generated and added to the hash table using the key and it is this random number that is returned. Using this method, the landscape is generated on a call-by-need basis. This is a very useful technique because in larger simulations it is likely that there is never going to be a need for every location. This method significantly reduces the set up cost of the simulation, and also makes is possible to run simulations that would otherwise take more time and processing power than available.

The class NKOrganisation is used to create the agents that traverse the landscape. These need to move to different locations on the landscape at each step of the simulation in order to improve their fitness. When an organisation is placed on a location, or when an organisation moves to a location, the fitness of the new location is calculated by the NKFitnessLandscape.

The adaptive walk method of the NKOrganisation is called on every tick of the simulation and is the method that allows the organisation to move over the landscape. The array nearestNeighbours (populated by the NKFitnessLandscape method getAllNeighbours()) contains all nearest neighbours of that location[1]. The next neighbour is then chosen from the set of nearestNeighbours (methods of choosing this can be seen in section 4.2.2 below). The fitness of the neighbouring location is found using the NKFitnessLandscape method getFitness(String key) as described above, this fitness is then compared to the fitness of the current location. If the fitness of the neighbouring location is greater than that of the current location, then the organisation moves to the neighbouring location. If the fitness of the current location is greater, then no move is made.

The organisation moves location by calling the moveTo(String key) method, this saves the new location into the organisation and resets all necessary variables for the new location.

The class NKDataCollector sets up the data collection for the NKModel using three classes; Fitness, NoFitterNeighbours and WaitTime. These calculate the maximum, average and minimum values of fitness, number of fitter neighbours and waiting time before the last move respectively over all organisations. This data can be shown on screen in three graphs or can be saved to file.

4.2.2 Extending the basic model

The following explains which extensions and parameters relate to which hypotheses made and also gives some of the more interesting implementation detail of the extensions to the model.

Next neighbour method

The parameter next_neighbour_method allows the user of the model to choose how the next neighbour to be looked at, is selected. The investigation into hypothesis 8 is also, therefore, enabled. The next neighbour to be looked at can be chosen in one of the following ways:

[1] An example of a set of nearest neighbours: if the current location is 010 (and A = 3) neighbouring locations are all locations that can result from changing the state of just one characteristic by plus or minus one so the neighbouring locations for this example are: “110”, “000”, “020” and “011”

- Taking the next neighbour from the set of neighbouring locations in a set order each time. The set of neighbouring locations is stored in an array (size M), this array can be accessed in order from index 0 to M - 1, incrementing the nextNeighbour variable each time a new neighbour is visited.
- Taking the next neighbour from the set of neighbouring locations randomly, with no memory of the last one chosen. A random number is generated between 0 and M - 1 and this location is used as the next neighbour.
- Taking the next neighbour from the set of neighbouring locations randomly, but with a memory of the ones already visited. This is implemented in the same way as the random selection with no memory. however, an ArrayList is created, this contains the locations already visited and is both checked and added to each tick, to ensure the same neighbour is not visited twice.

Realism of K

The parameter K_neighbours_or_random was created to allow the verification of one of Kauffman's claims; it is not important which characteristics are used for K (hypothesis 9). The parameter allows the selection of the method to be used when choosing the K characteristics for each N. If RANDOM is chosen then the K characteristics are drawn randomly from N (using the Repast uniform random number generator), however if NEIGHBOURS is chosen then K are selected as the nearest neighbours of N (default).

The parameter K_identical_or_random, settable at the start of the simulation, allows the investigation of hypotheses 10 and 11. The parameter reflects the choice of whether the number of K links for each characteristic should be; identical (default), selected from a uniform random distribution, or selected from a Gaussian random distribution. To implement this an array (K_array) is created, with length N, such that for each characteristic an individual K value is set in the array.

When the model parameter K_identical_or_random is set to IDENTICAL, then the integer in every cell of the K_array is K. When the model parameter is set to RANDOM, a random number is generated for each cell of the K_array in the range K plus or minus x. When the parameter is set to GAUSSIAN, a random Gaussian number is chosen and then mapped to the space K plus or minus x (such that (K + x) $<$ (N - 1) and (K - x) $\geq$ 0).

Realism of A

The parameter A_identical_or_random allows investigation into hypotheses 12 and 13. This parameter allows the user to select A to be either IDENTICAL, such that every characteristic has the same number of states, RANDOM such that every characteristic has a random number of states with A as the average number, or GAUSSIAN such that every characteristic has a number of states in the Gaussian curve (mapped to $0 - > 2A$ where the peak of the curve is at A).

Calculating fitness

Two parameters, fitness_method and fitness_method_averaging_weightings, allow the investigation of hypotheses 14 and 15 regarding changing the method that calculates fitness.

The parameter fitness_method allows the user to select either AVERAGE (default) or WEAKEST. Selecting AVERAGE means the fitness will be calculated by taking an average over the fitness of all characteristics where as selecting WEAKEST will mean that the lowest fitness will be taken.

The parameter fitness_method_averaging_weightings allows the user to choose (if they have chosen to use average fitness) whether the characteristics have IDENTICAL (default) weighting or RANDOM weighting when they are averaged. At the time, the author struggled to find a truly random way to implement this[2], however one was approximated using the following method. An array double weightings[N]; and a double total = 0.0; are defined by the program. A loop then iterates through each cell of the array, filling every cell (apart from the last) with a random number between total and 1.0 (the random number is then also added to total, such that total is a sum of the doubles in the array). On completion of the loop, the last place in the array is then filled by (1 - total), making the doubles in the array sum to 1.

Walking across the landscape

The inclusion of different walk types was incorporated via the parameter organisational_walk_type, allowing investigation into hypotheses 16 and 17. The parameter can be set to: ONE MUTANT NEIGHBOUR (default), FITTER DYNAMICS or GREEDY DYNAMICS.

The one mutant neighbour approach is the default and is the approach explained above, in the basic description. Greedy dynamics is very similar to the one-mutant neighbour method and therefore will not be touched on here, however fitter dynamics needed further implementation. For fitter dynamics, each element of the nearestNeighbours array is looked at in turn and the location of the highest neighbour is stored in an ArrayList. When another neighbour is found that is of higher fitness than the current one, the ArrayList is reset and the new location added. If, however, a location with exactly the same fitness as the current highest location is found, then it is added to the ArrayList. Then, after all neighbours have been tested, if there is more than one neighbour with the highest fitness (if there is more than one element in the ArrayList), a location can be chosen at random from among the fittest.

Jumping across the landscape: jump_J

To implement jumping (and verify hypothesis 18), the parameter jump_J was implemented such that when set, jumping across the landscape is enabled. The implementation of a jump, ensures that every tick, (after a local peak is found) a random new location is generated and checked against the current location to see if it is fitter. If the new location is not fitter, the organisation does not move and this is counted as an unsuccessful jump. If however the new location is fitter, then the moveTo(String location) method of the NKOrganisation is called. The organisation then moves to its new location and all variables are reset. This means that the organisation will begin walking once more and will not attempt to jump again until it reaches another local peak.

Introducing cost and setting limits

The parameters fitness_threshold, successful_jump_limit and maximum_jump_search should allow for the investigation of hypotheses 19-23. The implementation detail of these three parameters is as expected; for fitness_threshold a threshold it set and used within an if-statement to prevent jumping unless the new location improves fitness by more than this amount, for the other two parameters counters are used and incremented as appropriate.

[2] An alternative is discussed in section 8.3, further work.

Life and Death

Life and death (and thus the investigation into hypotheses 24-28) is facilitated through several user definable parameters.

A boolean life_and_death determines whether or not organisational life and death is to be included in this simulation and an if-statement separates the code used by this feature from all other code.

A double, life_and_death_threshold, defines the threshold at which organisations die. If the fitness of an organisation is less than the fittest organisation, minus the life_and_death_threshold, then that organisation is removed from the simulation. All organisations are part of an array list, agentList, and this is iterated through in order to access the organisations in the simulation. If an organisation is not part of the agentList, then it is no longer part of the simulation, therefore to simulate organisational death, an organisation is simply removed from the this ArrayList.

Lastly, the parameter new_organisation_method defines the method by which new organisations can be born. This can be one of three options: RANDOM NEW ORG, COPY OLD ORG, or BOTH. If RANDOM NEW ORG is chosen, a location is generated randomly and the organisation begins its life at this location. If COPY OLD ORG is selected, an organisation is selected randomly from among the current organisations, by using the random number generator to give a number in the range [0, agentList.size()] and the location of the organisation at this index is then used. If BOTH is selected, then the genetic load of the population is calculated and used to determine which of the above options to proceed with.

Chapter 5

The NK Model: simulation runs

The following chapter will first touch on the planning and preparation for the simulation runs and will then move on to discuss the results found after analysis had taken place.

5.1 Planning and preparation

The simulations for this stage were planned and carried out in two parts. First the ones to dock the model to Kauffman's model and then those to research the remaining hypotheses.

It was originally planned to complete ten runs for each simulation, however initial testing showed this to produce inconsistent results. Whilst results were largely consistent for higher values of N and K, for runs of the lower values, there was a lot of deviation. This can be shown through the results taken in the following sections, where, over 100 runs, there are vastly differing results for different simulations, even though exactly the same parameters are used. The standard deviation of for results at N = 20 starts at 0.05028 for K = 0 (with a mean of 0.6650) and decreases logarithmically as K increases, showing 0.00298 for K = 19 (with a mean of 0.63284). These findings highlight a necessity to take an average over a large number of runs, but also suggest that similar accuracy will be found, over a smaller number of runs, at higher values of N and K.

Due to these findings and mimicking Kauffman's research, this research runs 100 of each simulation and takes an average of these results. It was also planned that each of these simulations would run with 100 organisations, and the maximum, minimum and average fitness over these organisations would be captured.

In order to dock the model to Kauffman's Model, supporting evidence needed to be provided for hypotheses one through seven. To do this it was necessary to plan simulations that varied the parameters N and K. Values of N were looked at equal to 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, and 20 and values of K equal to 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16, 18, 19 up to and including K = N - 1 for each value of N[1].

Following this, in order to conduct the remaining research, the parameters in question were varied in a number of different set environments of N, K and A (in order to check that the same trends were found in

[1] It was initially thought to also run simulations for N = 50 and N = 100, however, having completed the first few N = 50 simulations, it was found that the 250 tick limit (allotted to all these simulation) was inadequate. Firstly it took two / three days to complete, and secondly it provided no significant results as most (90+) organisations were still walking at this point.

each environment). In general[2] four environments were decided upon (1) $N = 10$, $K = 5$ (2) $N = 10$, $K = 9$ (3) $N = 20$, $K = 10$ and (4) $N = 20$, $K = 19$ (in each environment A was set to two, apart from during the simulations where A was varied).

5.2 Docking the model to Kauffman's model

For the docking of this model to Kauffman's model, data was gathered from a series of different sizes of N and K in order to provide evidence to support or contradict hypotheses one to six[3]. Figure 5.2 shows fitness data for all these simulations and figure 5.1 shows the tick at which the last organisation stopped walking, for the same simulations.

K	N = 2	N = 3	N = 4	N = 5	N = 6	N = 7	N = 8	N = 9	N = 10	N = 15	N = 20
0	5	9	14	20	27	35	44	54	65	130	214
1	6	13	21	27	40	45	89	68	89	173	250
2		11	17	25	36	44	56	68	89	189	250
3			18	27	39	54	59	65	95	169	250
4				23	33	44	56	69	80	144	237
5					31	44	48	65	72	134	250
6						36	42	49	65	139	201
7							42	55	60	115	213
8								42	57	101	194
9									55	109	162
10										104	153
12										89	156
14										103	144
16											135
18											117
19											130

Figure 5.1: This table shows how walk times increase and decrease over different values of N and K. The value shown at each instance, is the tick on which the last organisations, within all simulations, stopped walking. The value of K that gave the longest walk length for each different N can be seen highlighted in yellow.

Hypothesis 1 predicted that as N increased and K stayed the same it would take longer to walk to the fitness optimum, because of the greater number of organisational locations to pass through. This is supported by the data gathered during these simulations.

For each value of K, as N increases, the average length of the walk also increases, as can be seen in figure 5.1. Taking the row K = 5 as an example, when N = 6 all organisations have stopped walking by tick 31, however when N is equal to 7, 8, 9, 10, 15 and 20 then, respectively, all organisations had stopped walking by ticks 44, 48, 65, 72, 134, and 250. This shows that as N increases in this set of simulation runs, the number of ticks reached before the last organisation stops walking also increases, as predicted.

[2]When testing next_neighbour_method, K_identical_or_random and K_neighbours_or_random the same values of N were used, but more values of K were tested.

[3]For results of hypothesis 7 see Realism of A, section 5.3.3

K	N = 2	N = 3	N = 4	N = 5	N = 6	N = 7	N = 8	N = 9	N = 10	N = 15	N = 20
0	0.61195	0.677967	0.672075	0.65168	0.683267	0.658714	0.663713	0.683967	0.66907	0.657007	0.664995
1	**0.677024**	**0.695342**	**0.69885**	**0.70349**	0.700483	0.706017	0.695477	0.702604	0.699842	0.696789	0.702872
2		0.688613	0.693565	0.696075	**0.70465**	**0.709536**	0.70271	0.701028	**0.705559**	0.703008	**0.715148**
3			0.688896	0.696088	0.701321	0.700788	**0.706656**	**0.707062**	0.70456	**0.705238**	0.707156
4				0.674022	0.692374	0.694043	0.695561	0.698897	0.701415	0.70071	0.705053
5					0.676222	0.680749	0.686404	0.690229	0.698263	0.698634	0.701297
6						0.669872	0.67545	0.68083	0.688197	0.692486	0.684277
7							0.665875	0.672941	0.676239	0.685923	0.679686
8								0.666107	0.668089	0.680242	0.674433
9									0.659999	0.674557	0.665167
10										0.668415	0.674433
12										0.656474	0.665167
14										0.643956	0.655483
16											0.645841
18											0.637115
19											0.632837

Figure 5.2: This table shows how fitness increases and decreases over different values of N and K. Fitness is always highest where K is low, but this maximal K increases gradually over time, as can be seen above, highlighted in yellow.

There is however one line of data that contradicts this hypothesis, this is where K = 1, here the longest walk peaks at 89 ticks for N = 8, and then goes back down again to 68 ticks for N = 9, before peaking again. This could be anomalous data, however it is not necessarily a strict contradiction to the hypothesis as all other data gathered is in support. More simulations should technically be done to confirm or invalidate these results, however there is not time for this at this juncture.

One other interesting feature here is that as K gets larger and N is fixed the walk peaks at a low K, this (although not part of this hypothesis) is a relevant feature and is related to the fact that the fitness of an organisation also peaks at a similar low K.

Hypothesis 2 predicted that as N increased and K stayed the same, the average fitness of the population would also stay the same. In general this hypothesis is supported. One example is where K = 0 (figure 5.2), here the fitness value normally stays between 0.65 and 0.68 (the only anomaly being at N = 2 where the fitness is significantly lower). A similar pattern to this can be seen in all lines of data, in some it looks as if the fitness could be gradually getting higher as N increases (for example K = 2, where there is initially an increase before the figures start fluctuating), but in others it looks completely random (for example K = 0).

Hypothesis 3 predicted that if K was set to zero, then when the end of the simulation was reached all organisations would have the same fitness and that when K was anything but zero, then when the end of the simulation was reached there would be different organisational fitnesses.

When the data was collected, the maximum, minimum and average fitnesses were also collected in every simulation (although the data show in figure 5.2 and figure 5.1 covers only averages). This hypothesis is unanimously supported by all data collected. Every one of the 100 simulations (for each of N = 2, 3, 4, 5, 6, 7, 8, 9, 10, 15 and 20) when run with K = 0 end with exactly the same organisational fitness for the maximum, minimum and average organisation. On top of this, every simulation run for K != 0 showed

very different maximum, minimum and average fitness values at the end of the simulations. For example see figure 5.3 where the two images show maximum, minimum and average fitness values for when K is equal to zero and when K is equal to one.

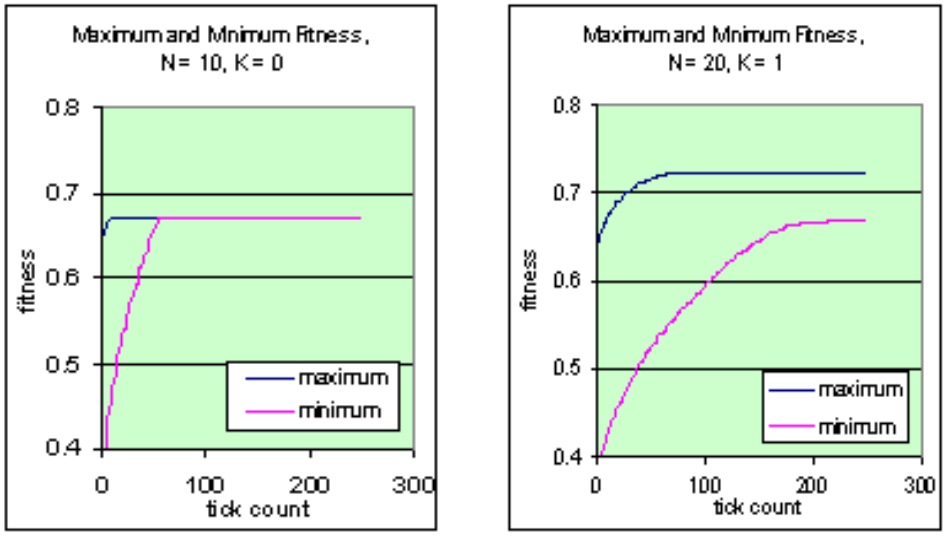

Figure 5.3: The two images show the maximum, minimum and average fitness for an N = 20 simulation. The image on the left is for K = 0 and it can be seen that at the end of the simulation all three values meet. The image on the right is for K = 1 and at the end of the simulation all three values are distinct.

Hypothesis 4 predicted that as N stayed the same and K increased, fitness would be maximal at a low value of K and **hypothesis 5** predicted that as N increased, the value of K that gave highest fitness would also increase, but not at the same rate as N. Hypothesis 4 is supported by the results found, as can be seen in figure 5.2, the highest fitness is always found at a low value of K. For example at N = 20 the highest value of K is still at K = 2. All the highest values of K have been highlighted in figure 5.2 and it can be seen that, as hypothesis 5 predicted, the values are gradually going up, if not steadily.

Hypothesis 6 predicted that the fitness of N would decreases towards 0.5 as K increased, because of the complexity catastrophe. This is also supported by the research here. This can be seen most clearly with the higher values of N. As soon as K goes past the low value at which it is maximal, the fitness begins to decrease, but in all the examples here it never gets to 0.5 but it appears as if it could be converging towards it. To establish this fully, more result should be collected, including results for higher values of N. This trend can also be seen in figure 5.4 where, as K increases (past its low maximal) in all three graphs, the fitness is lower for each higher K throughout the simulation. It can also be seen that, in fact, all locations in the landscape have lower fitness, not just the peaks.

5.3 Research using the NK Model

The following results show either supporting or contradictory evidence for the hypotheses made in section 4.1.3. Included in this chapter are examples of the results gathered, for further results see appendix E.

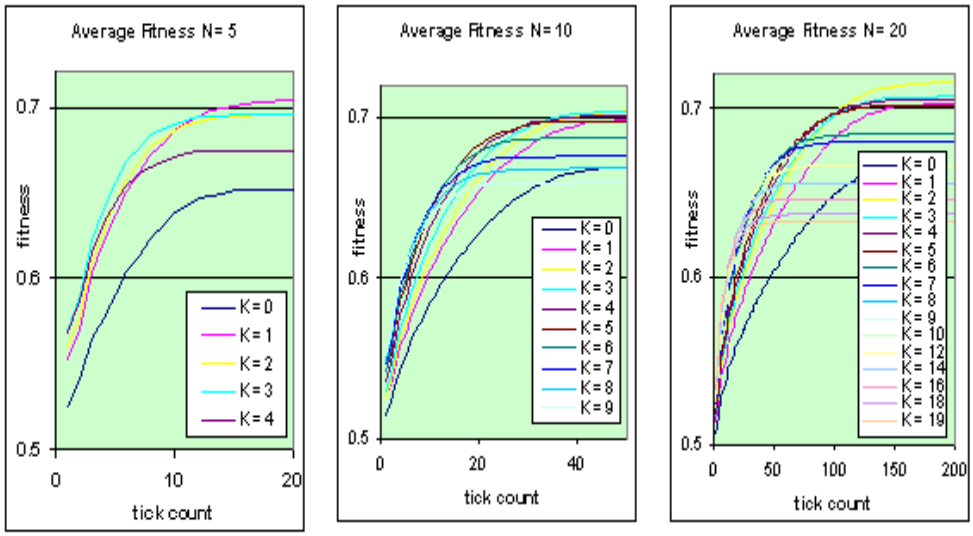

Figure 5.4: The three images show K increasing for N = 5, N = 10 and N = 20. In all three figures, the K with the highest fitness can be seen to be a low value of K (1, 3, 2 respectively) where as the very high K values (associated with a higher N) perform very badly.

5.3.1 Next neighbour method

The way in which the next neighbour is discovered is an important feature and results were gathered by changing K for N = 10 and N = 20.

Hypothesis 8 predicted that the method used to find the next neighbour, would not have a significant affect on the trends in the results gathered. Figure 5.5 shows a comparison between the possible different next neighbour methods for N = 20. As can be seen, hypothesis 8 is supported by the results in this figure, as all three methods give very similar end fitnesses. Also as predicted, when the neighbour is chosen randomly, but no memory is kept of previous choice, then this fitness takes longer to reach.

5.3.2 Realism of K

For K, we firstly set out to change which characteristics were chosen, either they were neighbours to N or they were chosen randomly from the set of N. We then moved on to looking at the size of K, first fixing K and then taking K from both a uniform random distribution and a normal random distribution.

Hypothesis 9 predicted that it would make no difference if the K characteristics, that affected N, were random or were neighbours of N, as Kauffman also predicted and later provided supporting evidence for. The results, as shown in figure 5.6 for N = 20, also support this hypothesis.

Hypothesis 10 and **11** predicted that when the number of dependencies K was a random number (either in a uniform distribution or in a normal distribution) the model results would be the same as when using K identically, because the average over a location would be the same. Evidence gathered supports both these hypotheses if talking about the trends discovered, but in actual fact the fitness itself is higher when

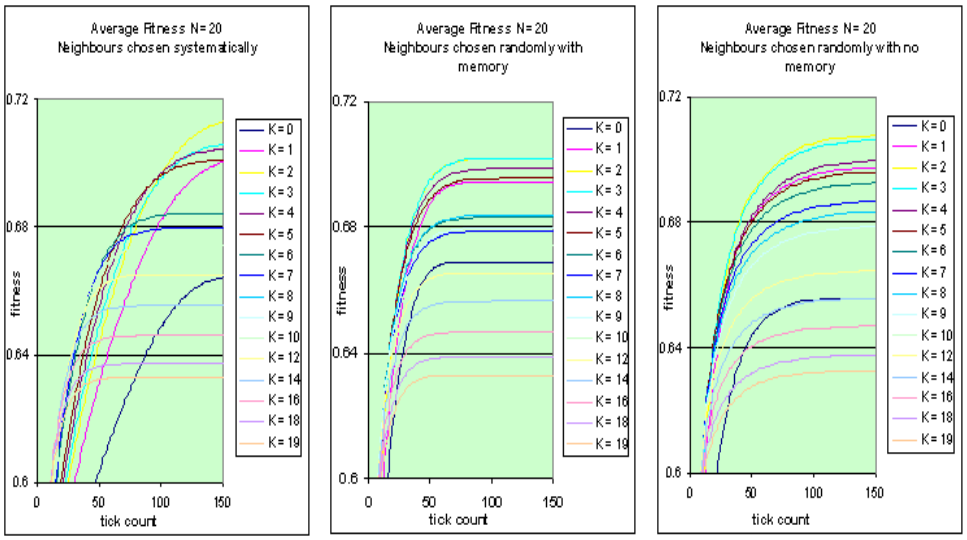

Figure 5.5: The three images show K increasing for N = 20. In the image on the far left the neighbours are chosen systematically. In the central image the neighbours are chosen randomly, but the same one is not chosen twice. In the image to the far right the neighbours are chosen completely randomly.

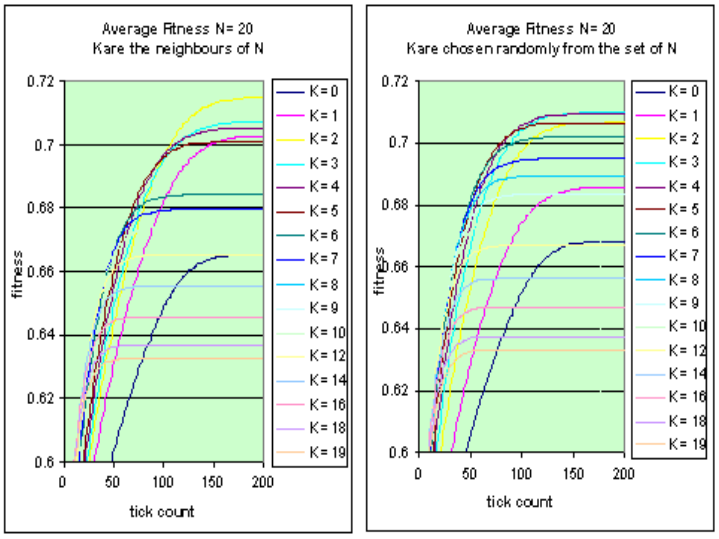

Figure 5.6: The two images show K increasing for N = 20, in the left image K are chosen as the neighbours to N and in the right image K are chosen randomly from the set of characteristics.

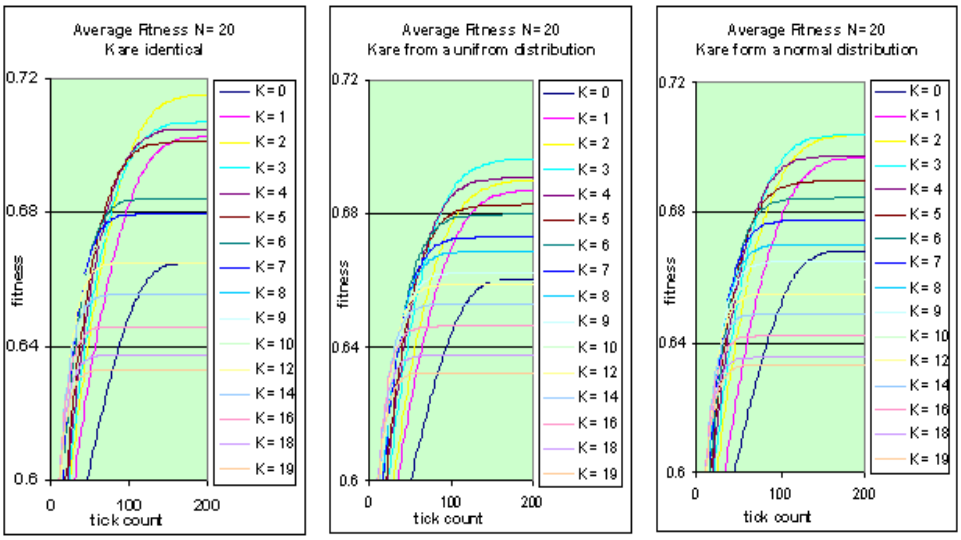

Figure 5.7: The three images show K increasing for N = 20, in the far left image every K is identical in one simulation run, in the central image the actual K used is random with an average set at the inputted K (using a uniform distribution), and in far right image K is also random (this time using a normal distribution).

K is identical.

Figure 5.7 shows these results, and whilst the figures are not exactly the same between the three graphs, the trends are and this is what is expected. In all three graphs the same low K has the highest fitness and also in all three, as K increases from that low K, the fitness decreases.

What is interesting however, is that the entire range of K values give higher fitness throughout, when K is identical. Next highest is when K is chosen from a normal distribution and lowest is when K is chosen from a uniform distribution. This shows that a higher fitness is gained, the closer K is to identical (choosing a random number from a normal distribution, ensures a number closer to the average than a uniform one, because of the bell shaped curve).

5.3.3 Realism of A

Kauffman (1993) set A to always be equal to two and did not investigate the affect of changing this. With respect to organisations this is very unrealistic and this is why one of the first further hypotheses made (hypothesis 7), was regarding this. The results from this are discussed first, followed by the results of changing A from fixed to random (taken from either a uniform random distribution or a normal one).

Hypothesis 7 predicted that as A (the number of states) increased, there would be a larger landscape with more locations and that it would therefore take a longer time to reach the fitness peak. Whilst this is supported, as can be seen from figure 5.8, where the number of organisations still walking at every tick is shown (and it can be seen that this is less for lower values of A), this is not the most interesting point to be made about changing A.

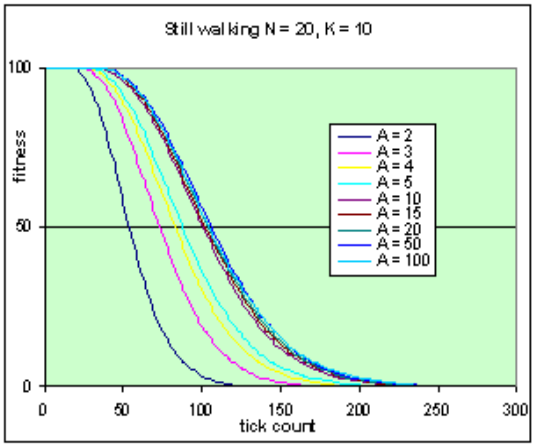

Figure 5.8: The above image shows number of organisations still walking at each tick, this is shown for N = 20 and K = 10 whilst A varies.

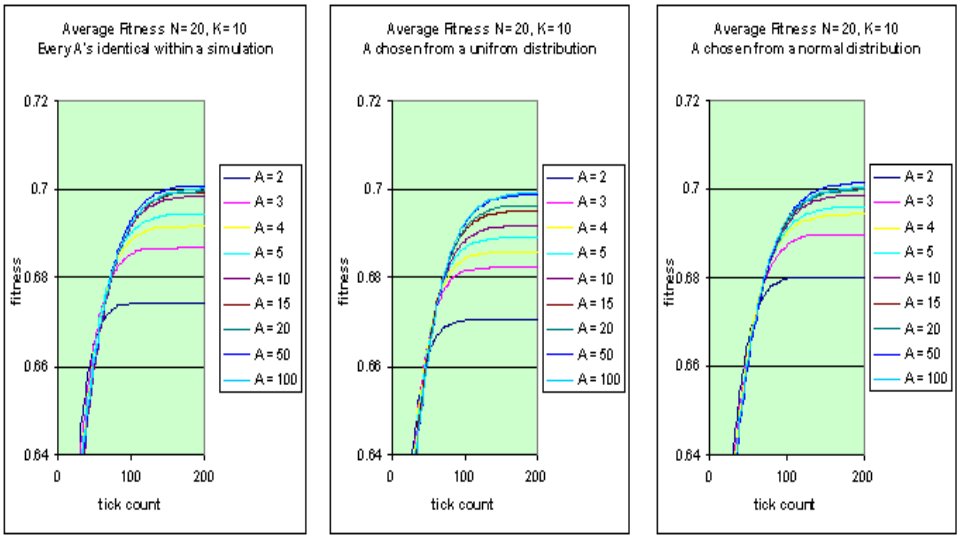

Figure 5.9: The three images show A increasing for N = 20, in the far left image every A is identical in one simulation run, in the central image the actual A used is random with an average set at the inputted A (using a uniform distribution), and in the far right image A is also random (this time using a normal distribution)

It was actually assumed, when the hypotheses were made, that the fitness of the highest peak would not be affected by increasing A and this is in fact not true. As can be seen in figure 5.9, the higher A gets, the greater the fitness of the peaks in the landscape, with the increase getting less significant the higher the A. The reason for this, is that because there are more locations, a greater expanse of the range [0,1] is used, meaning that there are more locations with a high fitness, and therefore more possibilities for a higher fitness peak (it is important to note here that there are not more peaks, the number of peaks is related to K, not to A). There are obviously also more lower locations on the landscape, however this does not affect the fitness of the higher peaks other than it takes longer to reach them as more lower locations must be crossed first.

Hypothesis 12 and **13** predicted that when A was chosen randomly with an average at A (either from a uniform random distribution or a normal random distribution) that this would not affect the model results. Supporting evidence was found for these hypotheses; figure 5.9 shows the original results on the left, following this with the results of when A is chosen from a uniform distribution and then from a normal distribution. As can be seen, these all show the same trends as A increases.

5.3.4 Calculating fitness

As discussed, three different methods of calculating fitness were investigated through the parameters fitness_method and fitness_method_average_weightings.

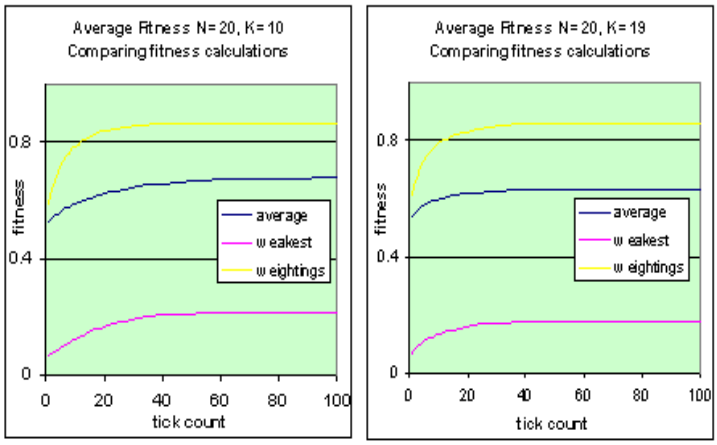

Figure 5.10: The two images show the three different methods of calculating fitness for N = 20 at two different values of K. In left hand image the three methods of calculating fitness are shown at K = 10 whilst in the right hand image the three methods of calculating fitness are shown at K = 19.

Hypothesis 14 predicted that when using the weakest fitness rather than an average, the overall fitness would be lower. To see supporting evidence for this, figure 5.10 shows a comparison of all three methods of fitness calculation and weakest is very clearly the lowest.

Hypothesis 15 predicted that when taking a weighted average, rather than a uniform average, there would be a higher than normal maximum fitness and a lower minimum fitness, but that the average fitness would remain the same. As can be seen by figure 5.10 this is not the case and in actual fact the over all fitness is higher. These results are significantly different from the predicted results and there is a similar reason

for this, as there was for why increasing A increased the fitness.

Weighting the characteristics does not just affect the peaks of the landscape, it affects every location, meaning that in fact *where* the peaks are in the landscape will change. As predicted, there will be some lower toughs because of this and some higher peaks. However, as stated when discussing the increase of A, having lower toughs does not affect the resulting fitness, only the length of time it takes to reach it, where as having higher peaks does increase the end fitness found.

5.3.5 Walking across the landscape

As discussed in the previous chapter there are in fact three different ways to walk over the landscape. The organisations can use either the one mutant change method, greedy dynamics or fitter dynamics.

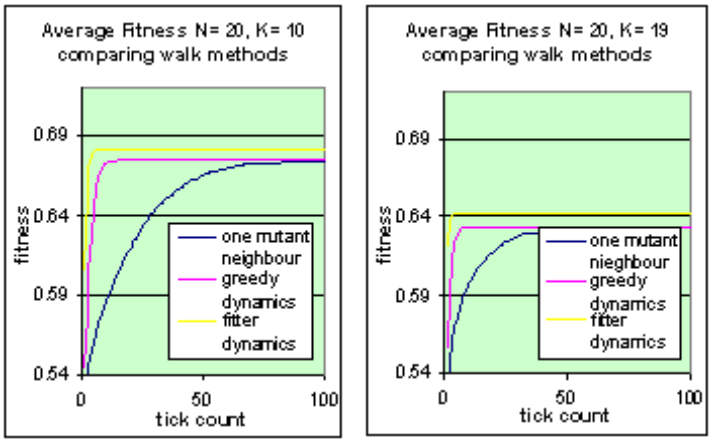

Figure 5.11: The two images show three different methods of walking across the landscape for N = 20 and two different values of K. The left hand image shows the three different methods of walking at K = 10 and the right hand image shows the three different methods of walking at K = 19.

Hypothesis 16 predicted that when using the greedy dynamics approach the final fitness location would be the same as when using the one-mutant change approach, however a stable state would be reached in less steps of the simulation. As can be seen when comparing the one mutant neighbour approach with greedy dynamics in figure 5.11, when using greedy dynamics organisations take a much shorter walk shown by the steeper curve towards the maximum. It can also be seen by looking at this figure, that a very similar average fitness is found by both, as the lines of one-mutant neighbour and greedy dynamics meet in both graphs.

Hypothesis 17 predicted that when using fitter dynamics the over all fitness of an organisation on the landscape would tend to be lower, due to the unexplored territory. In actual fact, as can be seen by figure 5.11, this is not the case; fitter dynamics (as the name itself suggested) does indeed find the fittest location on the landscape. This is likely to be because the steepest fitness peaks are also the highest, however more research would be necessary to support this claim.

5.3.6 Jumping across the landscape

Allowing an organisation to jump over the landscape gives that organisation more scope to improve its fitness, allowing the eventual convergence of all organisations towards the highest peak.

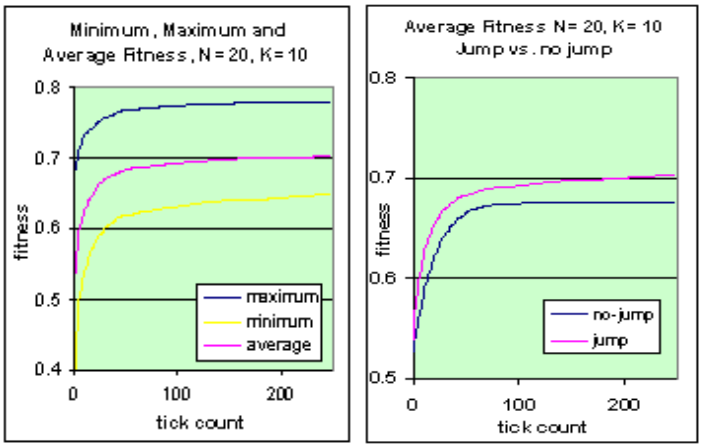

Figure 5.12: The graph to the left shows the maximum, minimum and average fitness of 100 organisation as they move and jump across the landscape for N = 20 and K = 10. The graph to the right however, shows the difference between the average fitness of a set of organisations that can jump across the landscape and a set of organisations that cannot.

Hypothesis 18 predicted that allowing organisations to jump over the landscape would increase the fitness of the population, such that given enough time the entire population would reach the highest fitness peak. As can be seen from the left graph of figure 5.12 in which the maximum, minimum and average fitnesses are shown, these values are converging towards each other. Unfortunately however, the simulations were not run for long enough to verify whether or not they actually meet. But, as is supported by the right hand graph in figure 5.12, when an organisation is allowed to jump across the landscape, it does reach a much higher fitness than if it was not able to jump.

5.3.7 Introducing cost and setting limits

This section looks firstly, at the fitness threshold, to determine the best threshold to set, to gain the highest fitness. Secondly the jumps limits are discussed in detail.

Hypothesis 19 predicted that increasing the fitness threshold would decrease the overall fitness of organisations on the landscape. And as can be seen in the supporting evidence of figure 5.13 this is very much the case. Even smaller fitness thresholds prevent the organisation from fully searching the landscape, therefore preventing them from gaining the highest fitness available.

Hypothesis 20 predicted that decreasing the number of successful jumps allowed would decrease the fitness of the over all population, this is supported by the evidence found, as can be seen from figure 5.14.

Hypothesis 21 predicted that increasing the number of characteristics (the size of the landscape), whilst keeping the number of successful jumps the same, would be more limiting and would give lower fitness

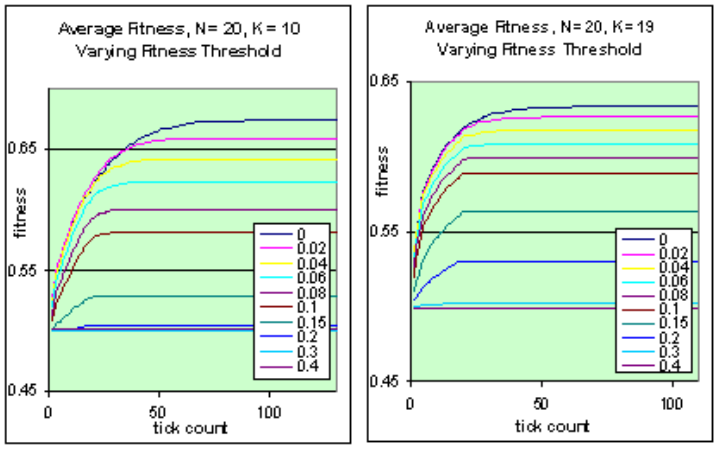

Figure 5.13: The two graphs show changing fitness threshold for N = 20 at two different values of K. The left hand images shows changing fitness threshold at K = 10 and the right hand image shows this for K = 19. As can be seen in both images, the best fitness threshold is in fact no fitness threshold, however having no threshold is unlikely within an organisation, due to expenditure.

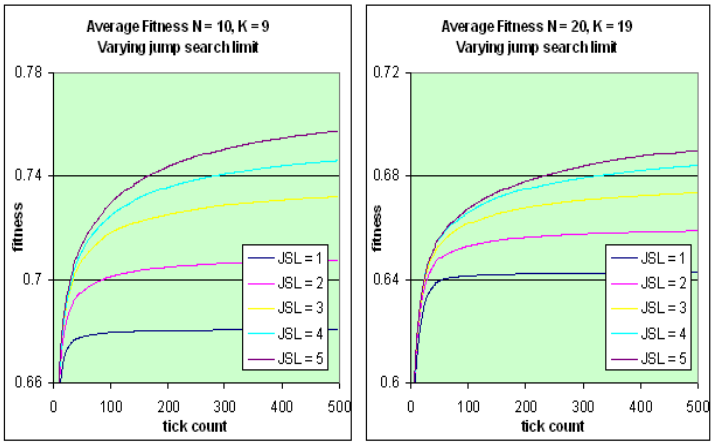

Figure 5.14: The two graphs show how changing the number of jumps allowed affects the fitness gained. The left hand graph shows this for N = 10, K = 9 and the right hand graph shows this for N = 20, K = 19.

for larger N. The evidence gathered also supports this hypothesis, as can be seen in figure 5.14.

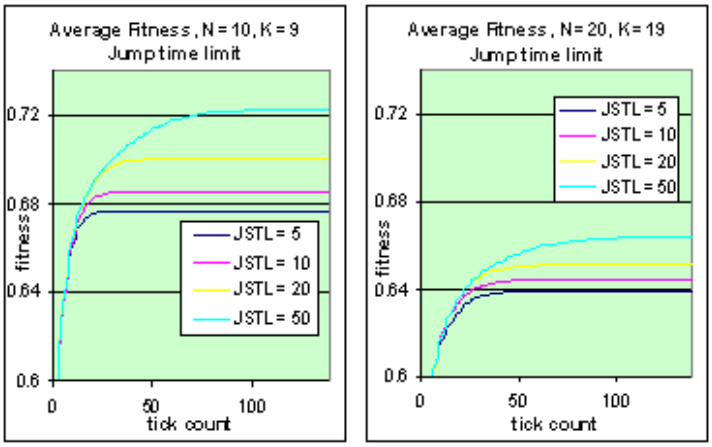

Figure 5.15: The two graphs show how changing the length of time an organisation is allowed to search for a new jump for affects the fitness gained. The left hand graph shows this for N = 10, K = 9 and the right hand graph shows this for N = 20, K = 19.

Hypothesis 22 predicted that as the amount of time an organisation can search for, increased, the fitness of the overall population would also increase. This is supported by the evidence gathered, as can be seen in figure 5.15 where there is a very noticeable difference between the results gathered under the different limits. The hypothesis also predicted that the increase would reach a limit when the fittest location in the landscape was found, unfortunately the limits set on the simulation did not allow this to be found, however this is not evidence that it does not exist.

Hypothesis 23 predicted that increasing the number of characteristics (increasing the size of the landscape), whilst keeping the time an organisation can search for the same, would be more limiting (giving lower fitness) for larger N. As can be seen in figure 5.15 this is certainly the case for the evidence gathered, the N = 10 graph (on the left) shows a much higher fitness than the N = 20.

5.3.8 Life and death of organisations

Organisations can both die and be brought to life and allowing the modelling of this in the simulation allows for the investigation of hypotheses 24 through 28.

Hypothesis 24 predicted that the smaller the death threshold (therefore the more death there is within the population) the longer it would take for organisations on the landscape to reach order. As can be seen in the three graphs in figure 5.16 this is supported by two of the three simulations. When new organisations copy old locations, this is not supported and order is in fact reached faster with smaller death thresholds. This is because all organisations immediately jump to the highest fitness. The results for the other two methods however, do support the hypothesis and smaller death thresholds take longer to find order (particularly when they are always chosen randomly).

Hypothesis 25 predicted firstly, that birth and death would give a higher fitness to the overall population, if the correct death threshold was found, and secondly, that this correct death threshold would neither be

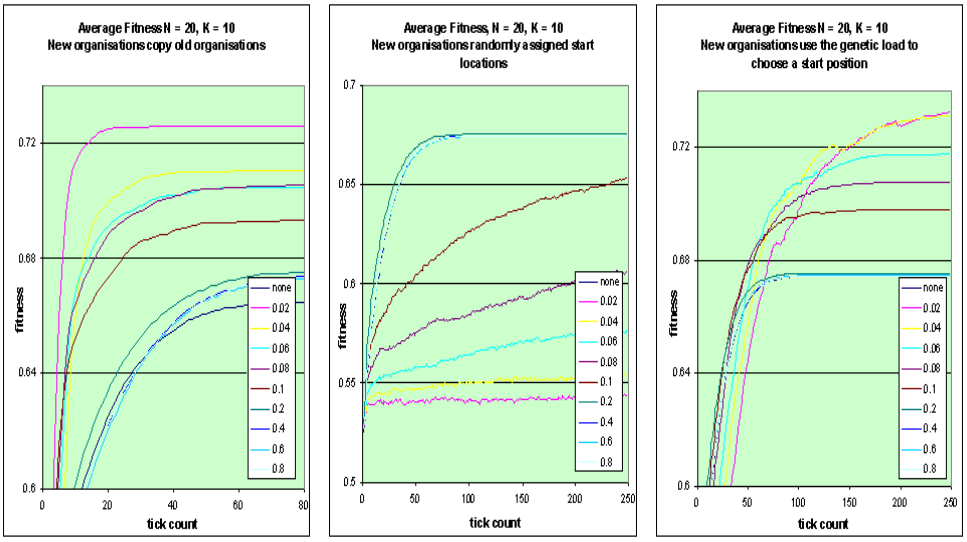

Figure 5.16: The three images show a varying death_threshold for the three different methods of new organisation creation. The far left image shows this varying death_threshold whilst a new organisation is created by copying the current location of an old organisation. The central image shows the varying death_threshold whilst a new organisation is created by randomly assigning an initial location. The far right shows the varying death_threshold whilst a new organisation is created by choosing (dependant on the genetic load of a population) either to copy an old organisation or to randomly choose a new location.

very large or very small. As can be seen in figure 5.16 both parts of this hypothesis are supported and the smaller fitness thresholds tend to give a higher fitness (apart from in the case of the randomly decided start location).

For randomly choosing a start location, the evidence is actually inconclusive. In the figure, it appears as if the fitness is lower using the smaller death thresholds, however it is obvious (due to the lack of curvature and lack of flattening in the series) that these organisations have not finished increasing their fitness, but have only been cut off by the end of the simulation. To decide whether or not this hypothesis is supported by this method, these simulations should be run again, for a longer period of time. Looking at the pattern shown in the above graph it might be best to capture up to 1000 ticks, to ensure all necessary data is gathered (this will not be done here).

Hypothesis 26 predicted that if new organisations always copied current organisations then the maximum fitness peak would be less likely to be found, as new territories would be left unexplored.

The evidence contradicts this hypothesis with regards to comparing this to having no life and death within a simulation, showing that in fact copying current locations finds a greater fitness peak than including no life and death. This shows that the unexplored areas do not have as much influence as the higher peaks found. With regards to comparing this to randomly choosing a start location (central image, figure 5.16), there is not enough evidence to either support or contradict this.

Hypothesis 27 predicted that firstly, when new organisations were randomly assigned locations, order would take longer to find than when they copied current locations. And secondly that a higher over all fitness would be found, because more new territory was covered. As can be seen in the central image of figure 5.16 this method definitely takes longer to find order and it has in fact not been found when the simulation ends. Because of this, it cannot be said whether or not this will eventually reach a higher fitness.

Hypothesis 28 predicted that using the genetic load to determine when to copy a current location, and when to create a random new location, would make order faster to find, but would still explore new territories when the population was at a generally low fitness. This hypothesis is supported, as can be seen in the far right image of figure 5.16, the fitness peaks are found in a much faster time frame to when organisations are given a random new location, but also the end fitness is higher than when only old organisations are copied.

Levinthal (1997) suggests that, through life and death of organisations, the selection process controls a single dominant peak. As can be seen, his selection process (facilitated by the genetic load, the far right image in figure 5.16), gives the greatest fitness over the time period studied. This suggests that the selection process is forcing more organisations to reach the higher fitness peak(s), supporting Levinthal's claims.

Chapter 6

The NKC Model: experimental hypotheses and design

This section will cover the two main tasks involved in planning the NKC Model experiments. Firstly the experimental hypotheses will be discussed in detail, looking at why these hypotheses were made and what stimulated research into this area. Secondly, the implementation of the NKC Model will be discussed, including a break down of the implementation of some of the trickier extensions.

6.1 Experimental hypothesis

As with the NK model, before starting to make hypotheses a short description will follow as a memory aid (again, for a more detailed description, please refer to section 2.3.3 of the literary review). Following this, hypotheses and related research will be discussed and planned. These should both verify the NKC Model and then use it in furthering the research into an organisations movements and adaptations in order to improve itself[1].

6.1.1 The NKC Model: a reminder

The NKC model is made up of a co-evolutionary set of species of organisation (this co-evolutionary set can contain two or more species) and a landscape for each species (specified by the parameters N, K and A). Whilst each species has its own distinct landscape, the landscape of each species is also linked (using the parameters X and C). The agent in this simulation is a co-evolutionary set of species, this set contains one of every type of species in the model and as the simulation progresses each species walks over its own landscape.

- N is the number of characteristics in a species of organisation
- K is the number of links that each characteristic has with other characteristics (within the same species)

[1] It should be bared in mind, that although the chapters of this dissertation are ordered in this way, the hypotheses made in this section were in fact made before the simulations for the NK Model were run

- A is the number of states each characteristic can be in
- S is the number of species of organisation in the model
- X is the number of links that each species of organisation has with other species
- C is the number of links that each characteristic has with other characteristics (within each of the X species it is linked to)
- f_1 is the function that calculates the fitness of a characteristic in a species of organisation
- f_2 is the function that calculates the fitness of a species of organisation

6.1.2 Hypotheses using the basic model

The basic NKC Model is the model exactly as it is described above, with no extensions. Whilst the research that could be done with the basic NK Model was only a repeat of past research, the NKC model has only, thus far, been used theoretically in management science.

The hypotheses made in this first section are still a repeat of Kauffman's research and they help show that the model designed and implemented here functions as Kauffman's NKC Model did. At the same time, however, they are researching the patterns of movement organisations might take in such a situation and this is a very different application to the one Kauffman was intending.

As such, hypotheses one to ten (below) attempt to align the model with Kauffman's conclusions. Hypotheses one to three are regarding his research into increasing S, the number of species [2] (Origins of Order pg 253) whilst four to ten are regarding co-evolving pairs of species (Origins of Order pg 249).

Changing the size of S

In order for the NKC model to realistically simulate organisations in their environment, the size of S must be variable as it is possible for there to be a varying number of organisations in a customer / supplier / competitor group. Conveniently, changing S is also something investigated by Kauffman and he comes to the following conclusions:

Hypothesis 1: as the number of species S increases, the waiting time to encounter Nash Equilibrium also increases. Nash Equilibrium is encountered when all species of organisation simultaneously mount a peak in their landscape; as the number of species in the environment increases, the likelihood of this occurring decreases.

Hypothesis 2: as the number of species S increases, the mean fitness of the co-evolving species decreases; this is due to the complexity catastrophe and the averaging over more characteristics. The C links to characteristics of other species, bring C more characteristics to average over for each new species added to the environment.

Hypothesis 3: as the number of species S increases the fluctuations in fitness of co-evolving species increases dramatically, this is due to the increased interactions which imply a higher chance of a species being displaced on its landscape by another species movement.

[2] Kauffman's research into different sizes of S is based on X being equal to S - 1 (such that each species is connected to every other species in the environment).

Investigating co-evolutionary pairs

Kauffman does much research into co-evolutionary pairs of species and how they are affected by changing the value of K (the number of links each organisational characteristic has with other characteristics within the same species). His results leed him to the following conclusions, which the model created in this research will attempt to verify:

Hypothesis 4: the fraction of co-evolving pairs that encounter Nash Equilibrium over a set number of generations (2500 in Kauffman's research) decrease as C increases. The higher C is, the more dependant each species of organisation is on the other, and thus the more likely it is that each move of one species will warp the landscape of the other, making it necessary for that organisation to move to maintain high fitness. Therefore, the higher C is, the more volatile the landscape and the less likely it is a species of organisation will find a stable state.

A good example of this occurring in the business world is a changing price. If a car sales man bases the price of his cars on the prices at the garage opposite, this price will probably not change too often (this is an example of a low C, C = 1). But, if he instead bases the price he sells cars at, not only on the price that they are sold at, but also on the make, colour, mileage, and quality of the cars opposite, it is likely that his pricing system will change more often (this is an example of a higher C, C = 5).

Hypothesis 5: when C is greater than one the fitness of each species of organisation is higher for those that reach Nash Equilibrium, than for those that do not. As highlighted when discussing the NK fitness landscape, order statistics show the average value of most fit location will be 2/3 (Kauffman 1993). An organisation that has reached Nash Equilibrium has more chance of meeting this fitness. Even if an organisation that has not reached Nash Equilibrium repeatedly passes through the location of highest fitness, it will have a lower average fitness than this maximum over any period of time.

To explain this, using a continuation of the pricing example mentioned above, co-evolving organisations that settle at a price (i.e. they reach this Nash Equilibrium) do better than ones that are still competing. This is because they do not loose out, either by selling their services or products too cheaply or by setting a price so high that customers are unwilling to pay and as such look else where.

Hypothesis 6: as C increases, the fitness of co-evolving pairs before they reach Nash Equilibrium decreases. The averaging affect causes the fitness of a location to converge towards 0.5 as the number of characteristics increase (this is due to both the complexity catastrophe, situation 2, see section 2.3.6 and also to the averaging affect over a greater number of species).

Hypothesis 7: when C is high (Kauffman uses C = 20), species of organisation with high K values do better than species will low K values. The higher K is, the better the species will do. This is because the higher K value holds each characteristic's fitness static - as other species in the co-evolutionary set change in value (as C change in value), K still stay the same. The greater the influence another species has over the fitness of the current species' location (i.e. the greater C is, relative to K) the more likely the fitness of the location is going to drastically change.

Hypothesis 8: when C is high (Kauffman uses C = 20), species with a low K do better against species with high K than they do against species of the same. This uses similar reasoning to that detailed above in hypothesis 7, species with a high K are not as easily affected by the changing fitness. This means that the species of high K will not change as often, in turn meaning that the species of low K are not then forced to move because of such a change.

As an example of this, if we have two species of K = 2 then if one changes location, it will warp the landscape of the other such that it too has to change. Predictably, as this second species changes location it then warps the landscape of the first species again, forcing a further change etc. In a second example

we have two different species one with K = 2 and one with K = 20. Whenever the K = 20 species changes fitness this will warp the landscape of the second species, such that it must then change location to compensate. However, the K = 20 species will probably not be affected enough by this change to force a second change. Due to this, the never ending cycle of changing fitness to compensate for each other (as seen in the first example) is avoided, allowing a higher fitness to be gained all round.

Hypothesis 9: when Nash Equilibrium is encountered the fitness of species with low K are higher than of species with high K. With high K, the affect of averaging across more characteristics brings the fitness of a species closer to the 0.5 fitness boundary. (This effect is also seen in the NK Model, though here, in the NKC Model, it only takes affect after Nash Equilibrium is found).

Hypothesis 10: when C is high, overall average fitness is higher when K is high, when C is low, overall average fitness is higher when K is low. Normally K is best when it is lower (as we discovered in the NK Model), however the stability a high K gives is more desirable when a high C is involved as without it, the high C will induce chaos.

Kauffman argues (because of what is suggested in hypothesis 7) that "the fitness in co-evolving systems would be enhanced were K able to adjust to match C, or more broadly, were K and C themselves evolving". A question stemming from this, therefore, is whether or not this would be possible within organisations. Could K be adjusted to compensate for an overly high C? If so, is this a good strategy to deal with the problem? Again, these are a stimulating questions, but not something answerable through use of this model.

Changing X

The final hypothesis in this section starts to extend Kauffman's research, whilst it utilises the model in its current form it looks into changing an aspect of the model Kauffman (1993) did not.

Hypothesis 11: as the number of co-evolving species S is fixed, at an arbitrary number, and the number of links between species, X, increases from zero up to S - 1 the system will stay in the state of chaos for a longer period of time and will also be less likely to find order. This is because more links induce higher interactions, meaning a species fitness is dependant on a greater number of other species and a characteristics fitness is dependant on a greater number of other characteristics. The more other species each species is connected too, the more likely it is that the never ending cycle, mentioned in hypothesis 8, will occur.

6.1.3 Hypotheses requiring extensions to the model

The following hypotheses move past Kauffman's research and look further into the way organisations move in relation to, and in conjunction with, each other. Adaptations are made to extensions from the NK Model, in order for them to also be used here and other extensions are also suggested.

Two very noticeable elements are left out of this research, that were included in the NK Model research, these are: jumping across the landscape and organisational life and death. There is no particular reason for this, other than a knowledge that not everything is possible in the time available. These two extensions were some of the later extension made in the NK Model and, being very separate from the rest of the model, were easier to leave out (this will be discussed in more detail in section 8.3 regarding further work).

Next neighbour method

The NKC Model has the same basic structure as the NK Model (including the landscape, the agents and the method of walking across the landscape) therefore, not surprising, an organisation still needs a method of choosing which neighbouring location to choose next. The three methods of choosing the next neighbour to look at are still applicable. These are[3]:

- Taking the next neighbour from the set of neighbouring locations in a set order each time.
- Taking the next neighbour from the set of neighbouring locations randomly, with no memory of the last one chosen.
- Taking the next neighbour from the set of neighbouring locations randomly, but with a memory of the ones already visited.

Hypothesis 12: as in the NK Model the method used to find the next neighbour will not have a significant affect on the trends in the results gathered, but this is due to different reasons than those discussed for the NK Model. As the locations in the landscape are randomly changing fitness, it will be random whether or not the next location looked at will be better than the current one whichever method is used to determine which neighbour to look at next. In addition to this, different species will likely follow different paths in the landscape, dependant on connected species' location, and again, this will not be affected by the next neighbour method chosen. For the reasons given, the method is technically random no matter which of the three options is used.

Realism of K

As mentioned in the NK Model it is more realistic to model K as varying for different characteristics. We can again hypotheses about the results this will show, this time in NKC model and it will be interesting to see if this has a different affect on organisational movement (although it is hypothesised that it will not).

Hypothesis 13: it makes no difference whether the K characteristics that affect N are random or are neighbours of N (as show by Kauffman for the NK Model).

Hypothesis 14: when the number of dependencies K is a random number, with an average of K, the model results will be the same as when using K identically (for the same reasons as mentioned in the NK Model description).

Hypothesis 15: when the number of dependencies K is a Gaussian number, mapped to the space plus or minus x (such that $(K + x) < (N - 1)$ and $(K - x) \geq 0$), the model results will again be the same as when using K identically (for the same reasons as mentioned in the NK Model description).

Realism of A

The same can be said for A, the number of states each characteristic can be in. It is more realistic to model this as varying for different characteristics in the model, and this too needs to be investigated to determine if changing this has the same affect in the NKC Model as in the NK Model.

Hypothesis 16: when A is chosen randomly with an average at A, rather than uniformly, this should not affect the model results (as described in the NK Model).

[3]For a more in depth discussion of these different methods and their applicability to organisations see section 4.1.3

Hypothesis 17: when the number of states A is a Gaussian number, mapped to the space $0 - > 2A$, the model results will also not be affected, because again the number of locations will not change dramatically as the average is still A.

Realism of X

X, the number of links between different species in the model, can be treated (and theorised about) in a similar way to K. In the basic model X does not vary between species, each species is connected to the same number of other species. Arguably, unlike K, at first glance this could be set to two and considered realistic, since each species probably has a supplier and a consumer, however in reality it is a lot more complex than this. Each species of organisation can potentially have a varying number of customer species and a varying number of supplier species, for example a supermarket must have a vast number of suppliers. Due to this it is definitely beneficial to research the affect of changing X in this way.

Hypothesis 18: when X is a random number, with an average at the inputted X, each species will be affected by a different number of other species. This means that some species in a co-evolutionary set will be more stable and others more volatile. There are two implications to this. Firstly, the co-evolutionary set will be influenced by the most volatile of the organisations, taking longer to reach order. Secondly, some species of organisation may reach order independently to others, see figure 6.1 (for a co-evolutionary set where species will always reach order at the same time) and figure 6.2 (for a co-evolutionary set where some species will reach order independently of others).

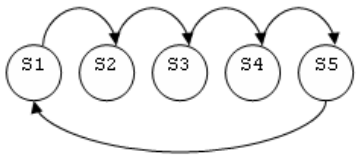

Figure 6.1: An example co-evolutionary set structure where all species will (if they reach order) reach it at the same time

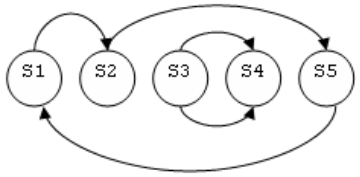

Figure 6.2: An example co-evolutionary set structure where different species will (if they reach order) reach it at two different times.

Hypothesis 19: when X is a Gaussian number, mapped to the space plus or minus z (such that (X + z) $<$ (S - 1) and (X - z) $\geq$ 0), again each species will be affected by a different number of other species, meaning some will be more stable and some more volatile. However, due to the Gaussian bell shaped curve, a randomly chosen X will generally be closer to the defined X in a Gaussian distribution than in a uniformly random distribution. This means that although the same trends will show, as with an X varying over a uniform distribution, the species of organisation will be affected even less.

Another potential research point is that in the same way as K, the X links between other species of organisation can either be taken as the neighbours of X, or can be taken randomly from the set of all X. With K, Kauffman brought evidence to support his hypotheses that which K used made no difference to the simulation, no such claim was made about X in the NKC Model, as Kauffman concentrates mainly on a fully connected co-evolutionary set (where X = S - 1).

Hypothesis 20: unlike K, where the choice of neighbouring characteristics or random characteristics will not affect the results of the model, the choice of X will have an affect (assuming that X $<$ S - 1, i.e. the species are not fully connected). If X is always taken as the neighbouring species, all species will be linked together through each other (even if X = 1). As such, if one species changes its location and thus

its fitness, this has the potential to change the fitness of EVERY other species in the landscape, meaning that no species show signs of order until all species do (e.g. figure 6.1). However, if X is random it is possible that some species will be linked separately from others, meaning distinct groups of species within a co-evolutionary set will co-evolve separately from the rest. In this instance one set might reach order and the other not e.g. in figure 6.2 S_3 and S_4 will reach order separately from S_1, S_2 and S_5 because they are not joined together. This could NEVER happen if only neighbours were used as X.

Realism of C

In the same way, C can also be theorised about similarly to K, however the properties of C are more similar to K than those of X. This is because C is also a link between different characteristics, just as K is. C, however, is between characteristics of different organisations rather than characteristics of the same organisation.

Hypothesis 21: when C is a random number, with an average at the inputted C, the model results will, as with K, not be greatly affected and there is a very logical reason for this. When C is random, for some characteristics within a species there will be more links to the characteristics of one species X_1 than to characteristics of another species X_2. This means that some characteristics will have a higher fitness that usual (as the value created has been averaged over fewer characteristics) where as others will have fitness closer to 0.5 (as the value created has been averaged over a higher number of characteristics). But, as some characteristics will have a higher fitness than normal and some a lower fitness than normal this will average out. This will also have less affect the higher X is, because there will be more species of organisation, and therefore more values of C for this to be averaged over.

Hypothesis 22: when the number of links to other species C is a Gaussian number, mapped to the space plus or minus z (such that (C + z) $<$ (N - 1) and (C - z) $\geq$ 0), the model results will be affected in a similar way as a uniformly random distribution. The model will again not be greatly affected by this (as detailed above), however there will also be less affect due to the bell shape of the Gaussian distribution curve, as in most cases the size of C will be closer to the defined size of C (now the average for the distribution). Again the affect will also become less as X increases.

Calculating fitness

As with the NK Model there are three different ways of calculating fitness; taking a uniform average of characteristic fitness, taking a weighted average of characteristic fitness or taking the fitness of the weakest characteristic. It is important to research how these different methods affect the fitness of an organisation, both throughout its walk over the landscape and of the end location found. It will also be interesting to see how (and if) these results differ from those of the NK Model, although the hypotheses predict that the same trends will be shown.

Hypothesis 23: when using the weakest, rather than the average, the overall fitness will be lower (as detailed in Chapter 4 regarding the NK Model).

Hypothesis 24: when taking a weighted average, rather than a normal average, this will affect the fitness of the landscape. As it is random which characteristics are weighted and by how much, there will be some locations that come out with a higher than normal fitness and some with a lower, meaning there will be a greater range of different finesses values over the landscape. There will be a larger than normal maximum fitness and a lower minimum fitness, but the average fitness will stay the same.

Walking across the landscape

As in the NK Model, there are the three different ways of walking across the landscape; one mutant change, greedy dynamics and fitter dynamics and organisations have the potential to use any of the three suggested ways. Again the hypotheses predict that the results and trends found will be the same as in the NK Model, however it is important to establish this through gathering supporting evidence.

Hypothesis 25: when using greedy dynamics the results will be the same as when using one-mutant neighbour, but each step will take longer to perform and a stable state will be reached in less steps of the simulation (if such a stable state exists).

Hypothesis 26: when using fitter dynamics the over all fitness of an organisation on the landscape tends to be lower (as detailed in the NK Model description, Chapter 4).

Introducing cost and setting limits

One of the major drawbacks of the these models is their assumption that there is no cost to any move an organisation makes. The addition of a fitness threshold, such that an organisation cannot move to a new location unless it is fitter than the current location by more than the fitness threshold, introduces such a concept. An organisation will not make a (potentially costly) move over the landscape for a tiny gain, this can and should be modelled in the landscape and the differences between doing this in the NK model and the NKC model should be studied.

Hypothesis 27: increasing the fitness threshold will decrease the overall fitness of organisations on the landscape, not just by the small amount that the threshold represents but by a greater amount, by blocking off potential pathways that exist by going through that location. Because of these blocked off paths it will also mean that the co-evolutionary set will find order faster.

6.2 Implementation of the NKC Model

As with the NK Model, the following section begins with a basic model description and adds complexity in layers, in order to test each set of hypotheses made. The headings here match those in the previous section, indicating the hypotheses to be experimented through the given implementation. (For a detailed specification of the NKC model see Appendix C).

6.2.1 The basic model

The basic implementation of the model has five main parameters that can be inputted before the simulation begins. Details about each species of organisation are added through an XML file. (Additional parameters, added after the completion of the basic model, will be mentioned later in this section and a full detail, of the use of all parameters, can be seen in Appendix D):

- S_species: S is the number of species of organisation within the simulation, each species of organisation has its own N, K and A, as a separate landscape is created for each species. For each species the following are needed:
  - N_size_of: the number of characteristic of each organisation

  - K_size_of: the number of characteristics each characteristic depends on (from within the same species)
  - A_size_of: the number of states each characteristic can be in

  These details (N, K and A) are NOT inputted via Repast parameters at the beginning of the simulation because there is no way to include a variable number of parameters in this way. Instead they are inputted using an XML file (the name and path of which can be input as a parameter).

- X_species: each species of organisation is linked to X other species (a species of organisation can be linked to between 0 and S - 1 other species). This link implies that these X other species of organisation affect the fitness of the current organisation.

- C_links_between_species: as previously described each characteristic of an organisational location is affected by K other characteristics of that location. In the NKC model, each characteristic is also affected by C other characteristics of each of the X species it is linked to.

- organisations_no_of: as in the NK Model, this is the number of agents that are thrown onto the landscape at the beginning of the simulation (default is 100). However, unlike the NK Model, here the agent of the simulation is actually a co-evolutionary set of organisations, rather than an individual organisation. It is these co-evolutionary sets that are thrown onto the landscape at the beginning of the simulation, the organisations that are part of the co-evolutionary set walk their own landscapes, but the fitness of their landscape is affected by all other organisations in their co-evolutionary set.

- fitness_range_dp: the number of decimal places the fitness range covers i.e. 0.0 to 1.0 or 0.00 to 1.00 (default is 2 decimal places)

As in the NK Model, the model does not include much parameter validation, assuming the model will be used for research (mainly by the author) and therefore the researcher will have enough knowledge of the model to input reasonable parameters. Only the vital validation is implemented, including: checking K is less than N; checking X is less than S; checking the XML file has the same number of species of organisation specified within it, as detailed by the input parameter S_species (if any of these conditions fail the simulation run will be halted).

The model is made up of six classes:

- The class NKCModel (very similar to the class NKModel) is where the model is set up and each step of the simulation is run from.

- The class NKCFitnessLandscape (again having a similar role to the class NKFitnessLandscape) contains all methods to calculate and access the fitness of locations in the landscape. The main difference is the method by which fitness of a location is calculated.

- The class CoevolutionarySet is a new class for the NCK Model and is the agent class of the model. Every instance of a co-evolutionary set will contain one of each species of organisation. These species are linked by X and C such that they affect / warp each others landscapes.

- The class NKCOrganisation is very similar to the class NKOrganisation, whilst not being the main agent class, the organisation within the co-evolutionary set can be thought of as an agent within another agent. As in the NK Model this class stores all methods for moving an organisation over its own landscape.

- The class XMLReader reads in the XML_file and parses it, converting the details contained in the file into an array of Species objects, which it then returns to the model class to be used in building the model.
- The class Species stores both the values of N, K and A, and the fitness landscape of a species of organisation.

Due to the similarities between the models, two abstract classes were introduced. NKModel and NKCModel already have an element of regularity, as they both extend the class SimModelImpl, however a regularity is forced upon the Organisation and the FitnessLandscape classes by the abstract classes AbstractOrganisation and AbstractFitnessLandscape. These two abstract classes allow the sharing of some methods and force the implementation of others, whilst keeping the main calculations of the implementations separate. In order to accomplish this, the NKOrganisation and NKFitnessLandscape were re-factored, moving certain methods to the abstract classes so they could be shared between the NK and the NKC Models.

The NKCModel provides the input parameter getter and setter methods, such that Repast can either create the user form from these (to be adjusted at the start of the simulation) or use a parameters file (for batch run). The class also provides the build and setup methods that set up the simulation. These start by initialising the data collector and then read the XML_file to import the specification for each species into the simulation. The landscape for each species is then set up and the co-evolutionary sets are created, giving them an organisation of each species which is placed at an initial location on its landscape. Lastly, as mentioned when describing the NK Model, the model class provides the pre-step, step and post-step methods, that run on every tick of the simulation, allowing a set of actions to occur for each agent (each co-evolutionary set). In this case, only the step() method is used and this is implemented such that the adaptiveWalk() method of the co-evolutionary set is called, this in turn calls the adaptiveWalk() method of every organisation that the co-evolutionary set has jurisdiction over.

The class NKCFitnessLandscape stores the landscape that an organisation of one species moves over during the simulation (such that a unique instance of NKCFitnessLandscape is created for each species within a simulation). The method getFitness(String key, String[] locations) returns the fitness of a species of organisation at the given location; in order for the method to do this, it must have the location of that species (String key) and the location of every other species it is connection to (String[] locations).

In exactly the same way as the NK Model, the fitness of the present location is an average of the fitness of every characteristic in that location, however the fitness of each characteristic is calculated slightly differently. As mentioned above, each characteristic, rather than just being dependant on itself and K other characteristics, is also dependant on C characteristics from each of the X species it is linked to.

During the re-factoring to include the abstract parent class, AbstractFitnessLandscape, the method characteristicFitness(int N, int K[ ]) was moved to the abstract class in order to be used by both NK and NKC landscapes. Therefore the same method is used here as is described in the NK Model implementation. The difference is that the array K[] must contain not only the states of N and all K, but also the sates of all C characteristics the current characteristic depends on, from each of the X species it is linked to, as showed in figure 6.3.

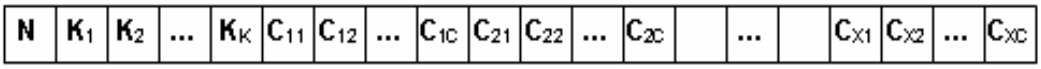

Figure 6.3: The structure of the array array_K in the NKC model

The CoevolutionarySet, the agent class of the simulation, contains an organisation of each species in the simulation, stored in an ArrayList. The class contains three methods; a method addSpecies() to allow the Model class to add species of organisations to the set on initialisation, a method adaptiveWalk() to iterate through and call the adaptive walk method of each species and a method makeLocationArray() to make an array containing the current location of every species in the co-evolutionary set (used by the adaptiveWalk() method).

The NKCOrganisation class is very similar to the NKOrgnaiztaion class and has the same methods moveTo() and adaptiveWalk() in order to walk over its landscape. The method moveTo() is extended from the abstract super class, whilst adaptiveWalk() is implemented from an abstract method. As in the NK Model, the adaptive walk is done using the one-mutant neighbour method, looking at each neighbouring location in turn and checking if it is of higher fitness. The walk process does differ from the NKOrganisation however, in that once it has looked at all locations it will not stop searching, as the NKOrganisation does, it will instead go round again. This is because, in the NKC Model, a change made by another species may have affected the fitness of either one of the locations already looked at (potentially making it of higher fitness than the current location) and / or the current location (potentially making it of lower fitness than surrounding locations).

The class NKCDataCollector collects the same detail as the NKDataCollector, but collects it over each organisation in a CoevolutioarySet.

6.2.2 Extending the basic model

The extended version of the NKC model has many of the same adaptations as the NK Model, because of these similarities, some of the descriptions will be brief here and the NK Model implementation will be refereed to.

Next neighbour method

The implementation that deals with this option, facilitating the investigation of hypothesis 10, was pulled into the AbstractOrganisation class and exactly the same implementation is used for both models.

Realism of K and A

Calling the method createKList() (moved from the NKFitnessLandscape to the AbstractFitnessLandscape) creates the array_K. It does this in the same way as in the NKC Model, using the parameters K_identical_or_random and K_neighbours_or_random. This enables the investigation of whether or not the distribution of K has an affect on the fitness of organisational species in the model (this should show the same results as the NK Model, as predicted in hypotheses 11 and 12). Also enabled by this piece of implementation, is the investigation into whether or not which K characteristics used to affect each N, have an affect on fitness (this should also show the same results as the NK Model, as predicted by hypothesis 13).

As in the NK Model, the parameter A_identical_or_random allows A to be either IDENTICAL, such that every characteristic has the same number of states, RANDOM such that every characteristic has a random number of states with A as the average, or GAUSSIAN such that every characteristic has a number of states in the Gaussian curve (mapped to $0 -> *2A$ where the peak of the curve is at A). This enables the research into hypotheses 14 and 15.

Realism of X and C

As mentioned, when discussing the implementation of K_identical_or_random, it is unlikely that every characteristic in a species will be affected by the same number of other characteristics, hypotheses 19 and 20 test changing this with regards to C. Realistically it is also seems unlikely that each species of organisation will be linked to the same number of other species, X, thus hypotheses 16 and 17 are intended to test this.

In order to implement these ideas, two arrays, the array_X[] and the array_C[][], were created (in a very similar way to the array_K). The array_X was created such that a set of X species could be randomly generated for each species of organisation. The array_C was created such that a C value could be set for each characteristic in relation to each of the X species it is linked with.

Introduction of the parameter X_identical_or_random allows the user of the model to specify whether all species should be linked to the same number of other species, or whether the number of species linked to should be assigned as random (with as average at X). When the model parameter is set to IDENTICAL, then the integer in every cell of the array_X is X. When the model parameter is set to RANDOM, a random number is generated for each cell of the array_X, in the range of X plus or minus z. When the parameter is set to GAUSSIAN, a random Gaussian number is chosen and then mapped to the space X plus or minus z. (Where z is an integer number, such that $(X + z) < (S - 1)$ and $(X - z) \geq 0$ where S is the number of species in the system).

The parameter C_identical_or_random allows the specification of whether the number of links between this each species is identical or random. Again, the choice of random distributions includes the uniform and the Gaussian distributions. When the model parameter for this is set to IDENTICAL, then the integer in every cell of the array_C is C. When the model parameter is set to RANDOM, a random number is generated for each cell of array_C, in the range of C plus or minus x. When the parameter is set to GAUSSIAN, a random Gaussian number is chosen and then mapped to the space C plus or minus x. (Where x is an integer number, such that $(C + x) < (N - 1)$ and $(C - x) \geq 0$ where N refers to the size of the species linked to NOT the current species).

X is the number of species that each other species depends upon and which X species are used is assigned by the method createXList() in the NKCFitnessLandscape. The question is; should these species be taken randomly from the set of species, or be taken as the neighbouring species and does this in fact make a difference? This will be researched by providing evidence to either support or contradict hypothesis 17. The changes necessary to the model in order to support this, can be implemented in the same way as they were the for the list of K. An input parameter is created, X_neighbours_or_random, whereby the user can set this to be either NEIGHBOURS or RANDOM.

Calculating fitness

Working out the fitness of a location in the simple model is done by retrieving the fitness of every characteristic in the location and taking an average. This fitness calculation can, in fact, be done in several ways, as in the NK Model. The parameter fitness_method, can either be AVERAGE: an average over all characteristics or WEAKEST: taking the weakest characteristic fitness (hypothesis 21). The fitness_method_average_weightings parameter, can be set to either IDENTICAL: whereby all characteristics have the same weighting, or WEIGHTINGS: where by each characteristic is assigned a random weighting (hypothesis 22).

Walking across the landscape: organisational_walk_type

As in the NK Model, the model has been extended to deal with three different types of walk across the landscape (to see a more detailed description of the types of walk see Chapter 4 regarding the NK Model). The three types of walk are: one mutant neighbour, greedy dynamics and fitter dynamics. This choice of walk type is implemented using the parameter organisational_walk_type, enabling hypotheses 23 and 24 to be researched.

Introducing cost and setting limits: fitness_threshold

Whilst none of the limits set on jumping (used in the NK Model) are relevant here, as jumping is not enabled, the fitness_threshold parameter was still introduced, allowing the investigation of hypothesis 25.

Chapter 7

The NKC Model: simulation runs

The following chapter will first touch on the planning and preparation for the simulation runs and will then move on to discuss the results found after analysis had taken place.

7.1 Planning and preparation

When the planning for this set of simulation began, two assumptions were made; firstly it was assumed that (as in the NK model) each simulation would need to be run 100 times and an average taken, secondly it was assumed that the fitness calculation would again be taken to two decimal places. After running a few tests (aiming only to estimate timings), these assumptions were found to be naive.

It was seen, due to the length of time each simulation took, that running 100 of each would be impossible considering the time available for the project. Each simulation takes one to three hours to complete, meaning 100 simulations would take 100-300 hours (four to twelve days). Consulting Kauffman (1993) it is unclear how many runs he did for each simulation, however looking at the results gathered (for example; Origins of Order, figure 6.2, page 246) the results are very varied, therefore it seems that they are not averaged. Also when species with the same parameters compete (as happens in most of the simulation done here) it is inappropriate to take an average, as this would bring results of competing species closer together, making the trends and differences unclear.

During this testing period it was also seen that taking a fitness calculation to two decimal places was impossible due to memory consumption. Even running simulations to one decimal place, as done by Kauffman, requires vast (and unavailable) amounts of memory. Because of this it was necessary to take fitness values to be either 0 or 1, rather than a double in the range [0,1], as integers take a lot less memory to store. Using integers tended to take 80% memory on the 512MB servers used, whilst any attempt to run the model using doubles as fitness values caused memory consumption to exceed 512MB and the program to thrash to virtual memory, greatly hindering performance. Whilst there are consequences to decreasing accuracy, after running more tests, it was seen that this did not in fact alter the trends found[1].

Following this initial testing, it was decided that results would be taken to the lesser accuracy described above and they would be taken over ten runs (for 100 co-evolutionary sets per run). Results of the ten runs conducted would then be compared for anomalies and one of the ten runs would be used for analysis.

[1] This is confirmed by section 7.2 where we dock this model to Kauffman's model.

Simulations were again planned in two sections, first the simulations to dock the model to Kauffman's model and then the simulations regarding further research.

The simulations to dock the model to Kauffman's model included three different sets of simulation runs. Firstly, simulations were planned to vary S (where X = S - 1), these set S to 2, 4, 6, 8 and 10. Secondly, simulations were planned to vary X for S = 5 and S = 10 with X equal to 1, 2 & 4 and 1, 2, 4, 6, & 8 respectively. Thirdly, simulations to vary K and C were planned, mimicking Kauffman's simulations (Origins of Order, figure 6.4, page 248). Each of the eight simulation was to be conducted over two species, with the same N but varying K and over three different values of C (1, 5 and 10); meaning 24 simulations in total.

Simulations to run further research were also planned, and it was known when planning that there would not be time to conduct all research desired. Because of this, a realistic stopping point had to be identified during this stage. It was therefore decided that only the following research would be conducted:

- X_identical_or_random: tests were planned to repeat the simulations that changed X but, with X_identical_or_random set to RANDOM and then set to GAUSSIAN (allowing the investigation of hypotheses 18 and 19).
- X_neighbours_or_random: tests were planned to repeat the simulations that changed X but, with X_neighbours_or_random set to RANDOM (allowing the investigation of hypothesis 20).
- C_identical_or_random: tests were planned to repeat the simulations that changed C but, with C_identical_or_random set to RANDOM and then set to GAUSSIAN (allowing the investigation of hypotheses 21 and 22).

7.2 Docking the model to Kauffman's model

This section details the results from the docking of this model to Kauffman's model, in order to accomplish this, data was first gathered to provide supporting evidence for hypotheses one to ten. Firstly, increasing S will be discussed and the affects this has on the involved species analysed and compared to Kaufman's results (hypotheses one to three). Secondly, co-evolutionary pairs of species will be investigated in depth, looking at both varying the K of the competing species and performing all tests at three different values of C (hypotheses four to ten). The graphs shown within the text below are examples of the results found, for complete results refer to appendix F.

7.2.1 Changing the size of S

The size of S was changed within a set environment, in order to facilitate the comparison of results. Two notable aspects of this environment were that firstly, in every simulation, X was set to the maximum (S - 1), such that all species were connected. Secondly, every species added to the co-evolutionary set had the same parameters for N, K and A (N = 10, K = 5 and A = 2).

Hypothesis 1 predicted that as the number of species S increased, the waiting time to encounter Nash Equilibrium would also increase. As can be seen by figure 7.1 the results gathered here support this, when S was equal four the stable state was encountered at an earlier tick than when S was equal to ten (the stable state being where the series flattens).

Hypothesis 2 predicted that as the number of species S increased, the mean fitness of the co-evolving species would decrease. This can also be seen to be supported by the results shown in figure 7.1; the end

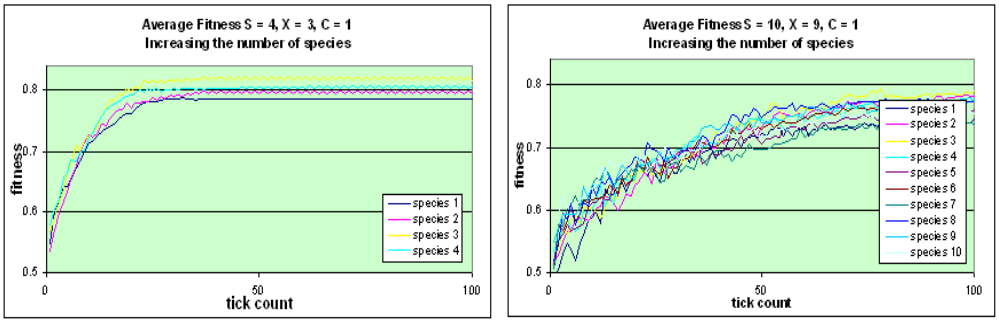

Figure 7.1: The two images show sets of fully connected, co-evolving species. The left image shows this for four species of organisation, where as the right hand image shows this for ten species. As Kauffman claimed, for the higher values of S, Nash Equilibrium takes longer to reach, overall fitness is lower and there are more fluctuations in fitness.

fitness found, and in fact the fitness all through the simulation, is higher for the lower value of S.

Hypothesis 3 predicted that as the number of species S increased, the fluctuations in fitness of co-evolving species would also increase dramatically. Again this can be seen in figure 7.1, as the graph of S = 10 fluctuations in fitness much more than that of S = 4.

7.2.2 Investigating co-evolutionary pairs

Co-evolutionary pairs of species were looked at in order to establish supporting evidence for hypotheses 4 to 10. The same eight experiments were conducted for C equal to 1, 5 and 10 allowing the investigation into changing C. Throughout the experiments both species of organisation were given the same N and A (N was set to be 25 and A to be two), whilst K changed between species (as shown in table 7.1), this allowed the investigation into changing K.

Table 7.1: This table shows the tests conducted for co-evolutionary pairs of species, each test was carried out at S = 2, X = 1, N = 25 and A = 2. The tests were also carried out for three different values of C; C = 1, C = 5, and C = 10.

Test	Species 1	Species 2
1	K = 2	K = 2
2	K = 2	K = 5
3	K = 2	K = 10
4	K = 2	K = 20
5	K = 5	K = 5
6	K = 5	K = 10
7	K = 5	K = 20
8	K = 10	K = 20

Hypothesis 4 predicted that the fraction of co-evolving pairs that encountered Nash Equilibrium would decrease as C increased. As can be seen by the example in figure 7.2, this appears to be the case in the

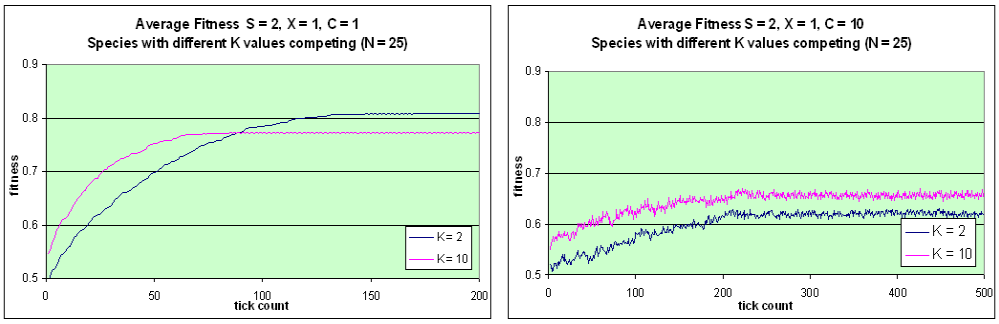

Figure 7.2: The two images show co-evolutionary pairs of species competing against each other. As C increases, fluctuation in fitness increase, the length of time taken to reach Nash Equilibrium increases and the overall fitness decreases.

supporting evidence, whilst these graphs represent an average over 100 co-evolutionary sets of species, it can be seen that the series fluctuates much more for the higher C, indicating more species that have not yet reached Nash Equilibrium.

Hypothesis 5 predicted that when C was greater than one, the fitness of the species of organisation would be higher for those that reached Nash Equilibrium, than for those that did not. Unfortunately, this is not clear from the data collected and data would need to be collected from all species within a simulation for this to be verified, not just from the species with maximum, minimum and average fitness.

Hypothesis 6 predicted that as C increased, the fitness of co-evolving pairs before they reached Nash Equilibrium would decrease. This is also supported by figure 7.2, as can be seen, the fitness is higher for C = 1 than it is for C = 10 and this trend occurs throughout the simulation, not just after Nash Equilibrium is reached.

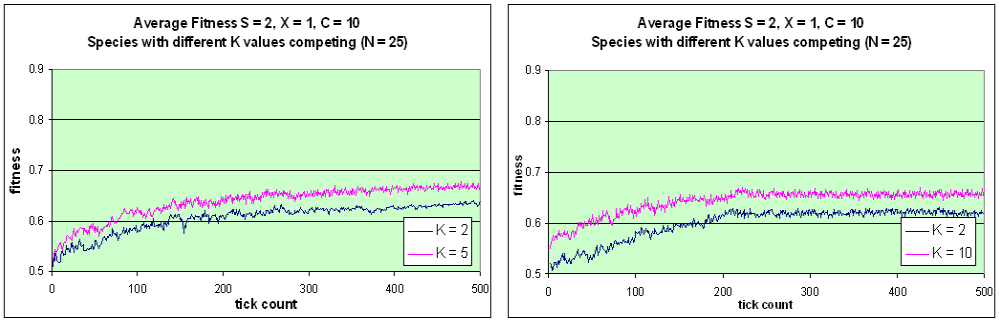

Figure 7.3: The two images show co-evolutionary pairs of species competing against each other. These two graphs are examples at a high C, that show a low K species being outperformed by a high K species.

Hypothesis 7 predicted that when C was high, species of organisation with high K values would do better than species will low K values. This seems contradictory to what we found in the NK Model, where a low K was always better, however two examples of a high K species outperforming a low K species can be seen in figure 7.3 and this is the pattern generally found for co-evolutionary pairs tested at C = 10.

There are, however, exceptions to this rule which occur at K = 2 vs. K = 20 and at K = 5 vs. K = 20 (as seen in the right hand image of figure 7.3 and in the appendix in figure F.4), here the low K value still does better. Kauffman tests his theories up to C = 20, (a test that takes too long to complete to be practical for our discussion) therefore this hypothesis cannot be said to be disproved by these results, as C is not as high as Kauffman intended it to be, when making this hypothesis.

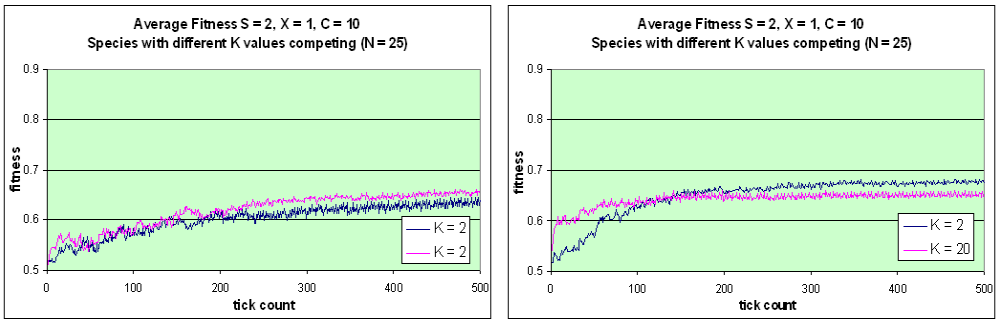

Figure 7.4: The two images show co-evolutionary pairs of species competing against each other. As can be seen, when a K = 2 species competes with a K = 20 species it does better than when it competes against another K = 2 species.

Hypothesis 8 predicted that when C was high, species with low K would do better against species with high K, than they would against species of the same. As can be seen in figure 7.4, the low K species (with K equal to two) does better when it faces a species with a high K, than it does when it faces a species of a similar K.

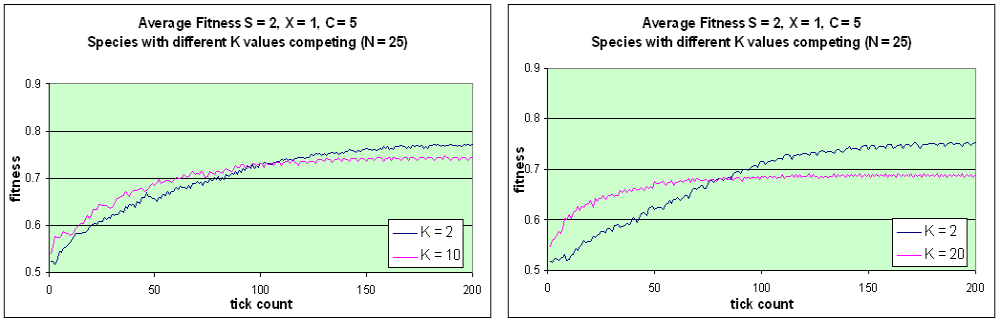

Figure 7.5: The two images show co-evolutionary pairs of species competing against each other. As can be seen, when Nash Equilibrium is encountered the fitness of species with low K is higher than the species with high K.

Hypothesis 9 predicted that when Nash Equilibrium was encountered, the fitness of species with low K would be higher than of species with high K. This pattern can be seen most clearly at C = 1 and C = 5, where Nash Equilibrium is more often reached (to better study these trends it is advised to refer to appendix F where all graphs for C = 1 and C = 5 can be found). Looking at the graphs for C = 5, figure 7.5, the pattern here shows the fitness of the low K levelling out at a higher fitness value than that of the high K.

Hypothesis 10 predicted that when C was high, overall average fitness would be highest when K was high, and that when C was low, overall average fitness would be highest when K was low. This can be seen most clearly in figure 7.2, where the same species types are shown competing firstly at C = 1 and secondly at C = 10. For C = 1, species K = 2 ends with a higher fitness, where as for C = 10, species K = 10 ends with a higher fitness; supporting our theory that a higher K can stabilise a species of organisation.

7.3 Research using the NKC Model

The following results show either supporting or contradictory evidence for selected hypotheses made in section 6.1.3. Included in this chapter are examples of the results gathered, for a full results listing see Appendix F.

7.3.1 Changing X

The first set of hypotheses to be discussed are those regarding changing the size of X for a fixed number of species. X was increased within two set environments; firstly for a co-evolutionary set of five species and secondly for a co-evolutionary set of ten species. The same trends were found in both environments, but the environment containing ten species is used for the example graphs here.

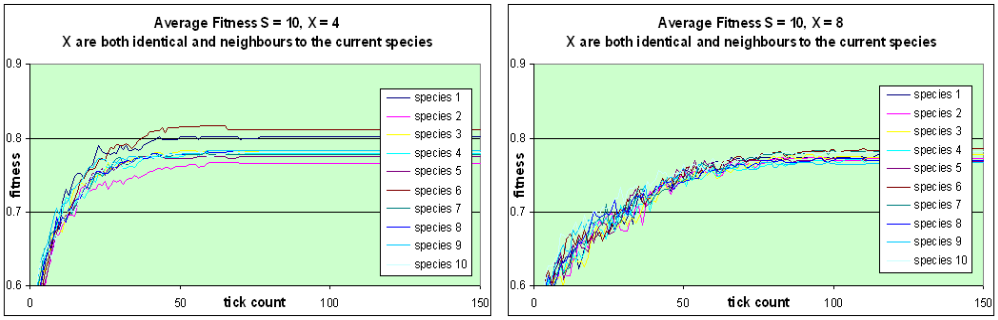

Figure 7.6: The two images show a simulation containing ten species of organisation. The graph to the left shows results for X = 4 and the graphs to the right shows results for X = 8. For the higher value of X Nash Equilibrium takes longer to find, there are more fluctuations in fitness and fitness is more consistently lower.

Hypothesis 11 predicted that if the number of co-evolving species S was fixed and the number of links between species X was increased, from zero up to S - 1, the system would stay in the state of chaos for a longer period of time and would also be less likely to find order. As can be seen from figure 7.6, whilst chaos does rain for longer, all species do still find Nash Equilibrium eventually. Whilst this is not supporting evidence, neither does it prove this hypothesis wrong, as if a higher number of species were experimented with a larger X could also be investigated. However, as seen by evidence in the following section, it seems more likely that either increasing C, or increasing C and X simultaneously, will prevent Nash Equilibrium from being reached.

7.3.2 Realism of X

Having looked at varying X within Kauffman's original specification, the results can then be compared to varying X when X is specified differently. The tests for changing X (detailed in the section above) were repeated, whilst the size of X was changed such that is would be random, with the averaged at the inputted X (rather than fixed for each species). This was done by choosing X from both a uniform distribution (in order to provide evidence for hypothesis 18) and from a normal distribution (in order to provide evidence for hypothesis 19).

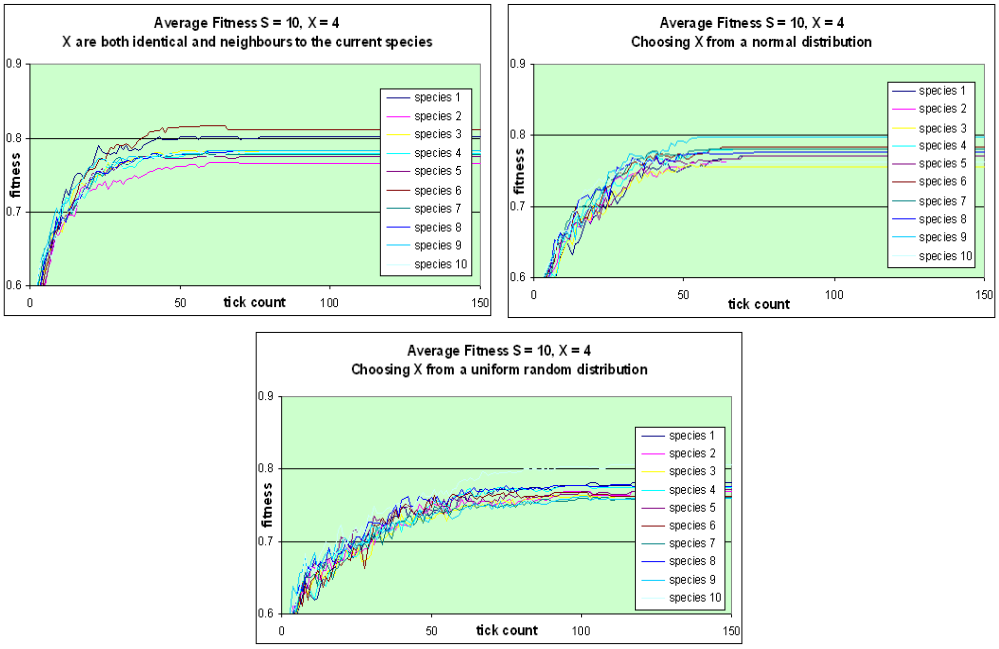

Figure 7.7: This figure compares results from S = 10, X = 4. The top left image shows a simulation where every species has the same number of X, the top right image shows a simulation where every species has a different number of X (using a normal random distribution) and the bottom image shows a simulation where every species has a different number of X (using a uniform random distribution).

Hypothesis 18 predicted that when X was a random number, with an average at the inputted X, each species would be affected by a different number of other species, meaning that some species in a co-evolutionary set would be more stable and others more volatile. It was predicted that there would be two implications of this. Firstly, the co-evolutionary set would be influenced by the most volatile of the organisations, taking longer to reach order. Secondly, some species of organisation would reach order independently of others.

As can be seen in figure 7.7, evidence gathered supports the first implication discussed, when X is random (as in the bottom graph of figure 7.7) the simulation does take longer to reach order. However, for the second implication of our hypothesis, we have no real supporting evidence, as whilst the graphs can be seen to show species reaching order at different times, this occurs however X is specified and is not dependant on X being chosen randomly (see appendix F figure F.5 for a large scale image of this).

Hypothesis 19 predicted that when X was a Gaussian number, again each species would be affected by a different number of other species, meaning some would be more stable and some more volatile. However, due to the Gaussian bell shaped curve, it also predicted that this would be a lot less noticeable than when X was chosen from a uniformly random distribution. This hypothesis is supported by the evidence gathered, as can be seen in figure 7.7, the results are very similar between the graphs and the Gaussian distribution (shown in the top right graph in figure 7.7) in fact shows results that appear to be half way between the other two.

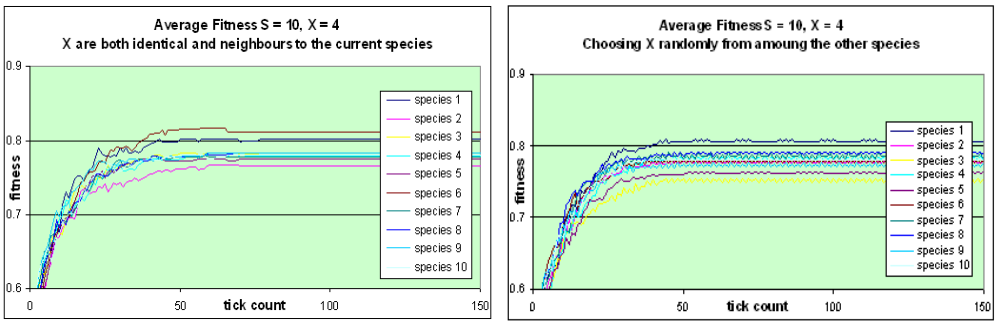

Figure 7.8: This figure compares results from S = 10, X = 4. The left hand image shows a simulation where the species chosen for X were the neighbouring species of S, where as the right hand images shows a simulation where the X species were chosen randomly from the set of all species.

Hypothesis 20 predicted that if the X species used were always taken as the neighbouring species, all species would find order together, where as if the X species used were chosen at random from the set of species, then different species would find order at different times. Unfortunately, this hypothesis has already been proved false, as stated previously, in all simulations different species may find order at different points in time, and this is not dependant on the specification of X.

It appears that despite what was predicted, the only clear difference is that if X are neighbours, order is not as stable as when C are chosen randomly (there are more fluctuations). This may be because of the structure of the network created; by linking only to the closest neighbours we effectively create a small world network, where as by linking randomly to different species we create a random density network. If, as a species of organisation, you are always linked to your neighbours and they are linked to their neighbours etc. there is a constant to which species affect which and you and your neighbour will (for the most part) be affected by the same other species. Where as if, as a species of organisation, you are linked randomly to different species and every species you are linked to is also linked randomly, the X links will not coincide in the same way.

7.3.3 Realism of C

Lastly, simulations were run to investigate the links between species, C. In order to establish a base by which to compare other results, simulations were run with C fixed between species, these were run at S = 5, X = 2 for varying sizes of C (2, 4, 6, 8 and 10). Following this, the same simulations were run twice more, but with C_identical_or_random set to RANDOM (in order to provide evidence for hypothesis 21) and then to GAUSSIAN (in order to provide evidence for hypothesis 22).

Hypothesis 21 and **22** predicted that when C was a random number, with an average at the inputted C, the

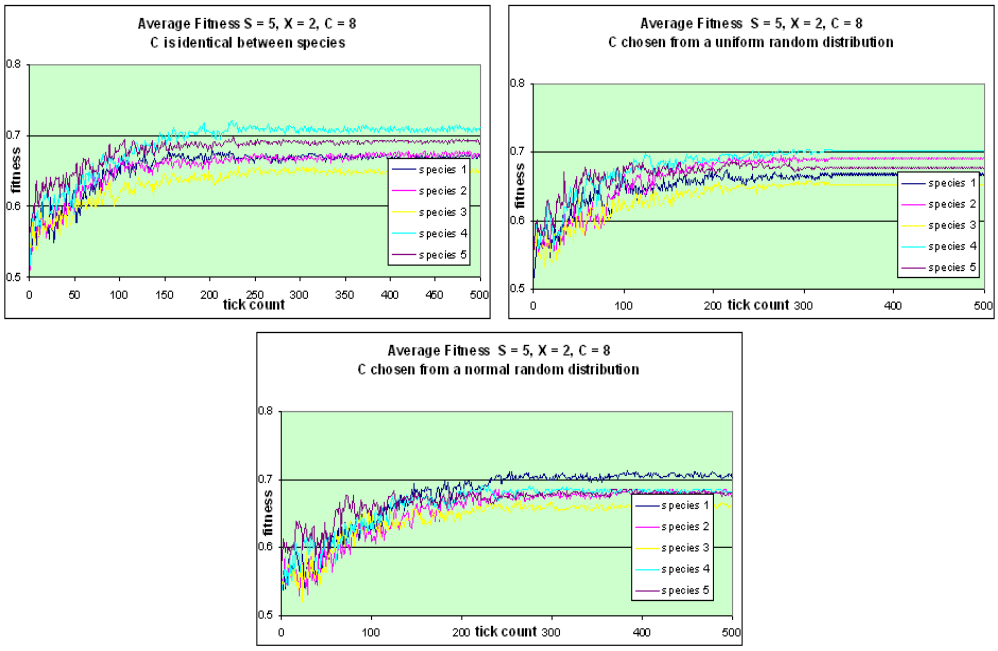

Figure 7.9: This figure compares results from S = 5, X = 2 and C = 8. The top left image shows a simulation where the size of C is fixed over all species, where as the other two images show simulations where C varies between species. The top right image shows a simulation where the size of C was chosen from a uniform random distribution and the bottom image shows a simulation where the size of C was chosen from a normal random distribution.

model results would, as with K, not be greatly affected. As can be seen from figure 7.9 this is supported by the evidence gathered. No matter whether C is identical, or is chosen from a uniform distribution (hypothesis 21) or normal distribution (hypothesis 22), the same trends are shown. Although, as found in the NK Model when studying K, overall fitness is actually higher when C is identical, meaning that, again, fitness can be seen to decrease as further complexity is added to the model.

Chapter 8

Conclusion

This final section will attempt to bring the project to a close by drawing some conclusions with regards to the results gathered from both models. Following this, the project will end with a critique of the work carried out and a few suggestions for further research, including both research available using the current model and by extending it.

8.1 Conclusion

The two models researched are very different and yet at the same time very similar, and both are critical to future research in this area.

It can be noted that the NKC model is, relatively, a much more accurate simulation of how an organisation might move in reality. This is because it is situated in a more realistic environment of competitors, customers and suppliers. The NK model however, is much easier to run and to analyse the results for. The complexity of NKC model means that a machine with sufficient processing power and memory must be found, as it uses a great deal of both; vast amounts of memory (due to the size of the landscape that must be stored) and vast amounts of processing power (because of the number of species involved in the simulation). The results are also harder to analyse because there are more results from different organisations to sort through and compare.

The NK Model is useful for changing individual parameters and noting the affect on the organisation in question, before it is applied in the more realistic setting of the NKC. In the NKC Model, the individual affect of this parameter change may not be as apparent, or may not be as expected. Using both models in this way will help determine if the change affects the organisation in question and / or the other organisations in the simulation.

The results gathered in this research show many different ways the models can be adapted to suit specific organisations and their methods and practises. The research tests many methods of constructing the landscape, of walking over the landscape and of calculating the fitness of a location and shows both the affects of these changes and the relevance of these to an organisation. The hope is that the research conducted here will both add to current research and facilitate further research in this area.

8.2 Project critique

The following section should give a summary and a critique of the process undertaken during the course of this project and should highlight whether or not the original objectives of the project were successfully completed.

Let us begin with the literature review, which gives an in depth discussion of the available literature regarding both models introduced in this project. This includes both Kauffman's original description for use within biological research and other authors interpretations, adaptations and suggestions in their efforts to tailor the models suitability to organisations. The biggest challenge of this project, and therefore the greatest achievement, was gaining an understanding of these models and the way that they work. This was also one of the primary objectives of the project, as the continuation of the project rested on this; the models could not be constructed accurately without this solid understanding. The idea behind the models is very simplistic, however the way they are constructed and the way this is explained in the literature, is not quite as simple. Because of this, a vast amount of research and evolutionary prototyping was done in the process of creating a full understanding.

A particularly complex element to understand was the function that calculated the fitness of each characteristic of an organisation and how this could be random and yet influential. An understanding of this was achieved gradually, through analysing results of a working prototype. Masses of research into the literature was conducted to try and establish exactly how this worked. However, it turned out that implementing this in the prototype facilitated a better understanding with which to re-read Kauffman's original text, and interpret the authors intended meaning.

Having gained an understanding of the original models it was also necessary to appreciate how this could be transferred for use within management science. From the survey of past research done, and also by intuition, possible extensions to the models were drawn and hypotheses were constructed with regard to these. Many authors had different ideas concerning how to make the models more suitable for organisations and a further objective of the models presented here was to try to draw these ideas together and bring, into one model, everything that can be associated with how an organisation functions. This was perhaps the second biggest challenge, finding the suitable extensions, understanding the authors meaning and applying these to the specification and implementation of the models.

Next it was necessary to form ideas of how this could be represented as an agent based-model. A related objective of the project was to research into applicable agent-based toolkits and choose one for this project, based on the research conducted and the suitability of the toolkits in question. As can be seen in chapter 3 this was researched very thoroughly and the toolkit Repast was chosen. From here is was necessary to construct a plan of how to transform Kauffman's models into functional Repast models. This was in actual fact simplistic, because of the help given through Repast it was clear that it was necessary to implement a model and an agent. The model should clearly be the fitness landscape of Kaufman's model and the organisation, the agent.

During the above mentioned processes the two models were specified in detail including both the basic elements of both models and the extensions derived from past research. From these specifications (and also from a knowledge of how both agent-based system work, and how Repast works) an implementation was draw for the NK Model. The resulting model was tested to ensure it performed as expected; this was the stage at which the evolutionary prototyping was carried out. Continuous testing was done to check the model performed as was appropriate and when this was not the case, the literature was again referred to, a better understanding was gained and the specification for the model was adapted where necessary.

Following this, an implementation for the NKC Model was drawn. During this, the implementation for the NK Model was refactored such that some of the implementation could be shared between the models.

Having gained such an in depth understanding of the models the implementation of the NKC Model was not difficult to derive from its specification and from the already implemented applicable sections of the NK model. Because of this, the implementation of the NKC Model may not be seen as that great an achievement, however in the larger picture this implementation may be one of the first to be practically applied to organisations. In management science there is much theoretical talk of the NKC model, but no evidence to suggest it has been used practically. This makes the construction of this implementation and the research that was conducted using it, more of an achievement.

Finally, having researched, specified and implemented the two models they were then used in research. Simulations were planned and carried out, firstly for the NK model and secondly for the NKC model. Simulations were carried out primarily to dock the models to Kauffman's and following this, to research further the affect of the extensions made.

Once the models were implemented it was possible to test the simulation runs and estimate how long simulations would take and it was found that the simulations would take longer than initially thought. As planned, half the time available for the project had been spent on implementation and half the time was left for simulation runs. Had there been less time available for implementation it would not have been possible to accomplish as much, therefore this time split was not necessarily badly planned. Simulations could potentially have been started as implementation continued, however it was thought best to ensure implementation was accurate before they were begun.

At any rate, the problem was already apparent, there was not enough time to conduct all desired research, therefore a solution was needed, rather than a investigation into the cause. In order to combat this, and in order to achieve another of the primary objectives (the accumulation of a large set of results for both models), more CPU power was acquired from different sources. However despite this, it was also required to cut down the number of simulation runs. Unfortunately this meant that many of the larger simulations (N = 50, N = 100 etc.) where left out, as these would take too long, and the simulations to be run for the NKC model were also halved (as these again took much longer to run than the NK simulations).

A further sacrifice was also needed in order to run the NKC model; the number of decimal places the fitness calculation was taken to. This, however, was due to memory constraints rather than to time constraints. Whilst the NK Model was set to an accuracy of two decimal places, for the NKC Model this would mean both running the model over many more generations (ticks of the model) and using more memory than available. In order to combat this, the number of decimal places used for the fitness of each characteristic was dropped to zero (meaning the fitness of each characteristic in a species of organisation was either 0 or 1). This meant that whilst the same trends were found[1] they occurred in a fewer number of generations and to a lesser degree of accuracy. Possible ways to combat this are discussed in section 8.3.1.

Finally, once results were collected analysis was conducted. Due to the volume of results acquired, the data files were imported into Excel, and Visual Basic macros were created to place the results into graphs. The data collected for different parameters could then be easily compared, facilitating the final objective of this project; the comparison of data, and the collection of supporting and contradictory evidence for the hypotheses made.

[1] Verified by the testing that docks this model to Kauffman's model

8.3 Further work

Due to both the time constraints and the vast potential of this project there is still much not covered. This is a very wide topic area and the further work suggested here does not nearly cover its extent.

Firstly there is still much research possible using the current models. This includes both research into all hypothesis made and not investigated, and also the thorough investigation of all available parameters. As mentioned in the relevant section, it was not possible to cover everything desired for the NKC Model, in fact almost none of the extensions made to the model (past Kauffman's research) were investigated. On top of this, there are also other parameters and implementations to the NK model that were made and not hypothesised about. Including the implementation of the communication networks of Lazer & Freidman (2005) (the reasoning behind not investigating this further will be stated later in this section).

Secondly, through initial background research, many ideas were generated that were never taken further, and there are many more possible extensions to the models that would be interesting to research in relation to management science. The bulk of these are for the NKC Model, this model was studied least by this research but, as argued in the conclusion, is potentially a better model for organisations than the NK.

The following section of this chapter discusses some of the many possible future research areas and the suggestions are split into four subsections. The first talks quickly about some implementation detail that could be improved, and the second moves on to discuss some further research applicable to both models. After this, the two models are looked at individually and further research possible for both is discussed in depth. The third subsection discusses one of the major drawbacks of the NK Model in relation to management science and provides some suggestions towards this. Whilst the fourth discusses a number of different possible extensions for the NKC Model.

8.3.1 Improving running speed and memory management of the model

Two of the major set backs of this research have been the speed at which the simulations run and the amount of RAM they require. For some of the longer NK Model simulations the runtime is nearly two days (to complete the 100 runs) and for the NKC Model one run can take up to three hours. Some of the larger model runs also required nearly 1GB of RAM, due to the size of the landscape required (this was an issue mainly the NKC Model, however the NK Model also had trouble when using the life_and_death option).

There have already been some attempts to speed up the processing the model. One being the creation of the landscape using a call-by-need method (rather than creating it entirely at the start of the simulation). Another attempt, was the use of some of the new hash table and array list implementations from Java 5, which allow the program to know the type of the object in the data structure such that casting was no longer necessary. As well as these, there are many other possible ways to improve the speed of a simulation, but some of these were discarded as they come at the expense of the object oriented structure of the program. For example, method calls and calls to objects take longer than if it were unnecessary to call the code from elsewhere, however not having these would make the code much less maintainable and much harder to read for potential future users of the model. Whilst these issues do exist, there are also other means that could be explored in order to try and improve simulation speed, for example loop optimisation.

With regards to the amount of memory used, there is a need for some form of memory management if larger simulations than those discussed in this project are to be run and if the accuracy of the NKC model is to be increased. One option is to change the toolkit used and to use a programming language

that does not consume as much memory as Java does, however this is not the only option. A second possibility is managing the use of memory, such that locations in the landscape are saved to hard disk after the organisation has passed through that area. Locations could then be brought back into RAM when another organisation moved into the area in question, to facilitate this, neighbouring locations in the landscape would need to be stored together in memory.

8.3.2 Both models: choosing a neighbour and weighting the fitness calculation

This sub-section will discuss two issues that affect both models. Firstly, the problems with choosing the next neighbour to be looked at and an additional approach to this. Secondly, the use of weightings in the fitness calculation and an alternative implementation.

The possible ways for a organisation to choose which neighbouring location to look at next were discussed in the hypotheses sections of both models. They included;

- Taking the next neighbour from the set of neighbouring locations in a set order each time.
- Taking the next neighbour from the set of neighbouring locations randomly, with no memory of the last one chosen.
- Taking the next neighbour from the set of neighbouring locations randomly, but with a memory of the ones already visited.

Having completed research regarding these different methods, and now looking back on this, all three options seem somewhat unrealistic with regards to how an organisation operates. Whilst organisations work strategically and are therefore likely to use a set approach / order, it seems more likely that different organisations would do this differently, as they have different priorities. Having said this, it still does not seem realistic to model this totally randomly, remembering previous locations looked at or not. Therefore a fourth method, that could also be of interest, is suggested here. This includes the construction of a set of different orders, in which to examine neighbouring locations. Having constructed this set, one of these is chosen at random to be used by each organisation and is then used by that organisation at every location. But, considering the lack of differentiation between the results found from the above three methods, it is debatable whether or not it is worth studying this factor further.

The fitness calculation can currently be calculated using three methods and one of these is a weighted average. However, the implementation of this weighted average does not actually give a truly random set of weightings and further research needs to be done using an improved method. One suggestion is to continue to use the weightings array, but rather than incrementally setting the weightings (such that each is smaller than the last), instead, assigning a random double to each cell, summing the array, and then dividing each element by this sum; this will achieve a more random set of weightings. It is important to note, however, that the method currently used in this research does not give incorrect results, only intensified ones.

8.3.3 The NK Model: adding communications

Within the NK model, most research avenues were discussed and explored, however there are two notable areas that could be of further interest. The issue brought to light by Lazer & Freidman (2005), with regards to communication networks between organisations, is the first item to be presented here. This follows nicely into a discussion about the links between life and death (Levinthal (1997)) and communication, which could also prove an interesting study.

Some of Lazer & Freidman (2005)'s ideas regarding networks were implemented within the NK Model during the project, however they were neither used in research nor discussed in the previous chapters. The reason for this, is that the implementation could be considered incomplete.

In the business world organisations do communicate, therefore the fact that the NK Model is devoid of any communication is unrealistic. However, as described, the past research of Lazer & Freidman (2005), when related to organisations, is just as unrealistic - organisations should not be able to completely copy each others location on the landscape, there is too much complexity involved. Despite this, some of Lazer & Freidman (2005)'s ideas and research can still be utilised. When applied to organisations, their research implies that organisations can gain fitness by communicating and learning from each other, rather than just walking over the landscape and learning through their environment.

Communications can currently be "turned on" in the model by setting the parameter comms to true, and a type of network can be selected using comms_network_type. The NKOrganisations are modelled as Nodes (extending the Node class) and there is an Edge class that connects linked nodes in the network. It is through these Edges (between nodes) that messages are left and picked up. The way communications are currently implemented allows for an organisation to know the exact position in the landscape of all other organisations it is connected to. As mentioned by Rivkin (2000), it is unlikely that any real organisation would be able to copy another so exactly, therefore a more realistic way of modelling this is needed.

There are a number of approaches possible, and research would need to be conducted to establish which method would be more realistic. This could be done by implementing and comparing the results of the different methods. A couple of possible approaches include:

- Rather than giving the whole location to connected organisations, instead giving the states of one or two random characteristics.
- Allowing agents to negotiate to share parts of their location, this would need to include some form of cost or information exchange.

Combining Levinthal's ideas of life and death, and Lazer and Friedman's ideas of organisations communicating, is not something that has been previously explored in research. It brings to light more questions about the use of communication networks and further research that can be done using them.

When combining communication networks and life and death there are three issues:

- How the death of an organisation affects the communication network: in the model implementation this is simple, all edges connected to this organisation are removed. In practise however, this will result in a fragmented network, which will deform further as time continues.
- How the birth of new organisations will affect the communications network: obviously new organisations cannot just take the place of old organisations (this would only be realistic if the linked organisations formed a client supplier link, and in this case - as they are organisations of the same form - they are more likely to be competitors). Therefore, there needs to be a method of adding new organisations to the network and this potentially should be different for different types of network. Whilst this was implemented in the current model, more thought potentially needs to be given to whether or not the ideas implemented are realistic and what can be done to make them more so. There are four types of network, as suggested by Lazer and Freidman, the network types and the implemented way of connecting a new node to these follow:
  - Fully Connected Networks: the new node is connected to every other node.

  - Random Density Networks: the new node is randomly assigned links in the network with the same density as given when the network was created.
  - Linear Networks and Small World Networks: links are added between this node and two other random nodes in the network.

- How networks change over time (not just through the birth and death of organisations): this is something that it is definitely important to research; organisations make new contacts and find new services, not just because the old ones disappear. Sometimes they are growing and need the new contacts, sometimes the old contacts give inadequate service and are left behind and sometimes new contacts materialise, that are not necessarily new organisations. This implementation of the model leaves the user to enter a percentage chance of the network changing over time, using the parameter comms_network_change_frequency, which can be anything from 0 to 100 (NB: if either the chance of change is set to 0, or if the network is fully connected, then it will not change over time).

The reasons these two items were not researched in this project are two fold. Firstly, time constraints, to ensure the results were valid this would need to be docked to Lazer & Freidman (2005)'s work before further research could be conducted, meaning less time spent conducting new research. Secondly, there was an element of indecision about how and what to communicate, and more research into practises of organisations in the real world would need to be performed to give a better idea of how to model this.

8.3.4 The NKC Model: a few additions

Due to the greater complexity of the NKC Model and the time it takes to complete one run of the model, (which is significantly greater than the time taken by the NK Model) not as much implementation was done. Therefore there are a significant number of extensions useful for further research.

Some of the NK Model implementations can be transferred straight into the NKC model e.g. jumping, however others cannot, for example communications and life and death cannot really be used in the same way.

The following is a short list of some of the more interesting further research areas:

- Jumping: allowing organisations to jump over the landscape in this model could be potentially disastrous, making order impossible to reach, depending on how connected co-evolving species are. Involving jumps is still very important however, for the same reasons they were included in the NK Model - most companies will not 'make do' with a sub-optimal peak, when they can see there are higher peaks available.

- Heterogeneity: species of organisation are grouped together within the model,meaning that all members of one species are seen as the same. It is possible, however, to introduce some form of heterogeneity. As mentioned in the literature review Kauffman (1993) suggests that "it is possible to extend the model to allow the population representing one species to be a cloud distributed over its landscape". This would enable research into the difference this has on results gained from the model.

- Life and Death: within the NK model we see organisations of the same species walking over the landscape and it is realistic to think of them as a group of organisations of which some might die and some might live. Within the NKC model we see species of organisations walking over the landscape, and whilst in some instances a species might potentially die, it is much more unlikely.

What is likely, however, is that if one species of organisation did die, it would take some of its co-evolutionary set with it e.g. if the car industry died out we would no longer need petrol stations or garages and this is something that is necessary to explore.

- Communications: whilst it is realistic, and makes sense, to think of organisations communicating with each other, this cannot be said of species of organisations. Whilst you can imagine an organisation communicating with another organisation, a species communication with another species is not likely. How communications between individual organisations might occur, seeing as we only model whole species, is still an open question.

- Differences between species: within previously suggested research, different species of organisation have been allowed different values of N, K and A. These are not, however, the only properties that could differ between organisations. They may also walk across the landscape using different methods e.g. one using fitter dynamics whilst the other uses greedy, alternatively they might have different limits e.g fitness_threshold.

- Species of organisation vs. individual organisations: one last suggestion is to use the NKC Model slightly differently to Kauffman and to imagine a co-evolutionary set as containing individual organisations, rather than species of organisation. In this way individual organisations might be modelled and their interactions with other organisations still taken into account. This would create more scope for communications networks and life and death within the NKC Model, but it would also add an extra layer of complexity.

All of the above examples could be hypothesised about, however extensions would need to be made to the implementation of the model in order for them to be explored.

Appendix A

Toolkit research

Table A.1: Initial Toolkit Analysis

Tool Kit	**Domain**	**User base and Support**	**Available**	**Language**
AgentSheets (AgentSheets 2006)(Repenning 2000)	Education	Y	N	Java / Visual AgenTalk
AScape (Institute 2000)	General	Y	Y	Java / Java
Bod-Mason (Bryson 2005)	Complex / Complete Agents (Bryson 2002)	Y	Y	Java / Python
Breve (Klein n.d.)	Spacial Simulation	Y	Y	Steve / Steve
CORMAS (The Green research unit 2006)	Resource Management	Y	Y	Smalltalk / Smalltalk
GoldSim (Group 2007)	Resource Management	Y	N	? / None
JACK (Group 2006)	Commercial Grade Resource Management	Y	Y	Java / JACK
Jade (Bellifemine et al. 2000)(Bianchi et al. 2006)	Assistant/Mobile Agents	Y	Y	Java / Java
JAM (Huber 2001)	BDI modelling	?	Y	Java / Java
JAS (Sonnessa et al. 2004)(Sonnessa 2004)	Social Science	N	Y	Java / Java
MadKit (Gutknecht et al. 2002)	Organisational Modelling	Y	Y	Java / Java, Scheme, Python
Mason (Balan et al. n.d.)	General	Y	Y	Java / Java
NetLogo (Welensky 1999)	Complex Systems	Y	Y	Java / Logo
Quicksilver (CollabNet 2006)	Social Science	N	Y	Java / Java
Repast (Altaweel et al. n.d.)	Social Science	Y	Y	Java / Java, .NET, Python
SDML (Wallis n.d.)	organisational Modelling	N	?	Smalltalk / SDML
SIM _AGENT(Poli & Sloman 1996)	AI and Robotics	Y	Y	PoPI 1 / PoPI1
SimPy (Team 2006)	Process Based	Y	Y	Python / Python
StarLogo (StarLogo 2000)	Education	Y	Y	Java / Logo
Swarm (SwarmUsers 2007)	Human and Social Science	Y	Y	? / Objective-C, Java
VSEit (Brassel n.d.)	Social Science	N	N	Java / Java
Zeus (Thompson 2000)	Rapid Design and Development	Y	Y	Java / ?

Table A.2: Detailed Toolkit Analysis

Toolkit	Userbase	Facilities for Collecting and Recording Data	Runtime Graphical User Interface
Repast	JavaDocs; tutorials; example models; FQA; mailing lists; developers mailing lists: developers, users, sourceforge; bug tracking system (Altaweel et al. n.d.)	result logging and graphical tool such as charts (including a variety e.g. sequence diagrams and histograms (Collier 2000)), network displays, QuickTime Movies and snapshots (Gao et al. 2003), 2D environment visualisation (Altaweel et al. n.d.), the probe (for agent inspection)	Repast has a graphical layer from which the model is started and parameters can be managed (Altaweel et al. n.d.)
Swarm	Documentation; user guide; FAQ; tutorials; example models; example applications; objective C documentation link; user contributed code available; one of the most widely known and used toolkits (SwarmUsers 2007)	has basic graphical tools including graphs and windows (SwarmUsers 2007)	there is no run time gui, swarm is started form the command line or from within the coding environment, any any runtime graphics must be programmed by the user (SwarmUsers 2007)
AScape	JavaDocs; mailing list (access denied); phone number for installation problems; scientific papers (Institute 2000)	graphs; QuickTime movies; agent and cell inspectors; charting and statistical tools (Parker 2001)	different GUI has different runtime views, dynamically changeable change/add/remove (Parker 2001)
NetLogo	Yahoo group 3248 users (NetLogo n.d.) (NetLogo 2007); forums; example models; user manual; FAQ; contacts; bug reporting (Welensky 1999)	graphs and charts; data is saved in a plot which can be exported to spreadsheet and database packages(Welensky 1999)	Agent monitors for inspecting and controlling agents at run time; runtime GUI; 2D and 3D graphics (Welensky 1999)
MadKit	User guide; designed agent examples; development guide; FAQ; bug tracking (0 messages); forums (13 messages), very few users	graphs and charts(Gutknecht et al. 2002)	the GUI has a list of available agents, a properties zone and there is an agent desktop working environment (looks very similar to windows) and an Agent/Group/Role organisational Model (Detlor & Serenko 2002) (Gutknecht et al. 2002)
Mason	Mailing list and archives; documentation; tutorials; example projects and models; scientific papers (Balan et al. n.d.)	downloadable extra packages for generating graphs and charts (Balan et al. n.d.)	can be run with or without GUI and can switch between during a run, very fast without GUI; different visualisation possible at the same time e.g. 2D and 3D(Balan et al. 2003)

Appendix B

Specification of the NK Model

A model is constructed of a fitness landscape of distinct locations and each location in the landscape has a fitness value (in the range 0.0 to 1.0). An organisation takes the fitness of the location it currently occupies, it starts at a random location and then moves over the landscape, continuously trying to improve its own fitness by moving to fitter locations.

B.1 Constructing the landscape

1. A fitness landscape should be constructed of all possible locations / organisation configurations. There are A^N possible configurations e.g. A = 2, N = 5, $2^5 = 32$.

2. A location is made up of a set of characteristics (N) that each organisation has

   a. COMPULSORY: It should be possible to specify the number of characteristics, N

3. A characteristic can have a number of states (A) and that characteristic can take any one of those states (e.g. if A = 2 then each characteristic can be in state 0 or state 1). Each different combination of sates of N is a different location on the landscape (e.g. if N = 2 and A = 2 the following locations exist: 00, 01, 10, 11).

   a. COMPULSORY: It should be possible to specify A the number of states

   b. OPTIONAL: It should be possible to specify whether A is identical (over every characteristic in a location) or random (with A as an average) over the different characteristics

4. The fitness of a location on the landscape (and therefore of any organisation situated at this location) is the average of the fitness of the characteristics in that location (point 6 explains how to calculate the fitness of one characteristic).

   a. OPTIONAL: It should be possible to specify whether we are calculating fitness by:

      i. Working out an average over the K + 1 characteristics (Kauffman (1993))

      ii. Taking the fitness of the weakest characteristic (McKelvey & Yuan (2004))

      iii. Taking a weighted average by randomly assigning characteristics different weightings (Solow et al. (1999))

5. The fitness of a characteristic is dependant on its own state and the state of K other characteristics that affect it. This fitness is worked out such that for every possible combination of the states of N and K a random fitness between 0.0 and 1.0 is assigned.

6. K can be defined by the following:

   a. K specifies a relationship between N_i and N_1, N_2, ..., N_K (where i $= 1, 2, ..., K$). This is a directed relationship i.e. just because N_i is affected by N_j this does not force N_j to be affected by N_j

   b. In every different location each characteristic N is affected by the same K characteristics

   c. COMPULSORY: It should be possible to specify the number of K characteristics that each characteristic is dependant on

   d. OPTIONAL: It should be possible to specify whether the size of K fixed for each characteristic or it varies between characteristics using K as an average (McKelvey & Yuan (2004))

   e. OPTIONAL: It should be possible to specify how K should be chosen from the set of all characteristics, K could either be: (Kauffman (1993))

      i. The nearest neighbours of N

      ii. Picked randomly from among the set of all characteristics

B.2 Traversing the landscape

1. An organisation progresses in its movement over the landscape at each time step and it should be possible to specify how they progress (Rivkin (2000)), either by:

   a. COMPULSORY: Walking over the landscape ()

   b. OPTIONAL: Walking and long jumping over the landscape

2. A walk may be taken by examining neighbouring locations. Each location has D = (A - 1)N one mutant neighbours, a one mutant neighbour is a neighbouring location that varies from the current one by the state of one characteristic.

   a. It should be possible to specify how to choose the next one mutant neighbour

      i. The neighbour is chosen from a list of all neighbours systematically (in the order that the neighbours appear in the list)

      ii. The neighbour is chosen from the list at random

   b. OPTIONAL: It should be possible to specify a threshold such that if the change gives under the threshold improvement the change will not be taken:

      i. If no threshold is specified any change may be taken

      ii. If a threshold is specified the change will only be taken if it gives improvement greater than the threshold (McKelvey & Yuan (2004))

   c. OPTIONAL: Alternative methods of calculating fitness include:

      i. Greedy Dynamics: each neighbour is looked at in turn and the first neighbour found that is fitter than the current location is moved to

      ii. Fitter Dynamics: all locations are examined and the fittest location is moved to (if there is more than one alternative one should be chosen at random).

3. A long jump may be taken when an organisation reaches and becomes trapped upon a local peak. In order to take a long jump a location will be selected at random, its fitness will be calculated and this new location will be moved to if the fitness is higher than the current location.
   - **a.** Long jumps are discovered by sending a clone to look at these locations, the original organisation doesn't commit until it knows there is a higher fitness level at the new location. If the current location under investigation isn't fitter then the clone jumps around the landscape (on following ticks) until a fitter variant is found
   - **b.** OPTIONAL: It should be possible to specify a maximum number of successful jumps the organisation can make
     - **i.** If a maximum isn't specified jumps continue indefinitely / until the simulation ends
     - **ii.** If a maximum is specified then once this number of jumps is reached the organisation will remain on the final peak
   - **c.** OPTIONAL: It should be possible to specify a maximum length of time the clone can search for
     - **i.** If a maximum isn't specified the search continues indefinitely / until the simulation ends
     - **ii.** If a maximum is specified then once this length of time is up the organisation will remain on the final peak reached
4. The fitness of the landscape is calculated as the average of a number of walks over the landscape.
   - **a.** A range of walks over each landscape must be taken into account e.g. 100
   - **b.** A range of landscapes must be taken into account e.g. 100

B.3 Optional extras to implement

1. It should be possible to turn on an option that will allow the birth and death of an organisation as specified by Levinthal (1997). This should be simulated over each time step, such that organisations can be born and can die but the over all number of organisations doesn't change.
   - **a.** At the end of each time step some organisations should die (dependant on the fitness of the organisation with respect to the fittest organisation).
     - **i.** It should be possible to specify the acceptable difference between the fittest and the least fit organisation.
   - **b.** For every organisation that died a new organisation must be born (to keep the numbers static) the organisation is thrown onto the landscape and can start in a position that:
     - **i.** Is a copy of a successful current organisation.
     - **ii.** Is randomly selected from all possible organisation start points.
     - **iii.** Is a mixture of both such that when organisations are doing well in general it is (i) and when they are not it is (ii) this should be calculated using the genetic load function (Levinthal (1997)).
2. It should be possible to turn on an option that will allow communication between organisations. Communication should occur at each time step and therefore there is the possibility of copying a more successful organisation communicated with (Lazer & Freidman (2005)).
   - **a.** At each time step communication will occur and if a fitter organisation is communicated with this organisation will be copied, if not a walk across the landscape will be possible.
     - **i.** It should NOT be possible to exactly copy another organisation

ii. There should be a way to specify how much of another organisations location on the landscape is communicated

b. It should be possible to specify which organisations can be communicated with by specifying a type of communications network:

i. Random density network

ii. Fully connected network

iii. Linear network

iv. Small world network

Appendix C

Specification of the NKC Model

C.1 Constructing the landscape

1. The model contains a number of different species of organisation (S) and a landscape for each of these species specified by an individual N, K and A for each species

   a. It should be possible to specify the number of species S

2. Each species is linked to X other species.

   a. In Kaufman's description of the model it is assumed that every species is connected to every other species but this is not necessarily the case in reality, therefore it should be possible to specify the number of X species one species is linked to

   b. It should be possible to specify how X are chosen:

      i. Each species is linked to its nearest neighbours (wrapping round if necessary)

      ii. Species are chosen randomly from the set of all species

3. As mentioned above each species of organisation has a different fitness landscape that it traverses, and as such it has a different value for N, K and A:

   a. It should be possible to specify the size of N for each species

   b. It should be possible to specify the size of A for each species

      i. OPTIONAL: It should be possible to specify whether the number of states (A) is identical across all characteristics or whether it should be chosen from a random distribution with the average at A

   c. It should be possible to specify the size of K for each species

      i. OPTIONAL: It should be possible whether the number of links between characteristics (K) is identical across all characteristics or whether it should be chosen from a random distribution with an average at K

      ii. OPTIONAL: It should be possible to define if the K dependencies are:

         (a) The nearest neighbours of N

         (b) Characteristics chosen at random form the set of all N

4. The fitness of a location in the landscape should be calculated, as before, as an average of every characteristic in that location.

   a. OPTIONAL: It should be possible to specify an alternative to this, and to decide whether we are calculating fitness by:

      i. Working out an average over the K + 1 characteristics (Kauffman (1993))

      ii. Taking the weakest (McKelvey & Yuan (2004))

      iii. Giving different characteristics different weightings rather than just taking the average (Solow et al. (1999))

5. The fitness of a characteristic is calculated in the same way as in the NK Model but not just with respect to K affecting characteristics (within the same species) but ALSO with respect to C affecting characteristics (from each X species linked to). This means that the same location on the landscape may have a different fitness for different species of organisation dependant on their co-evolving partners.

   b. COMPULSORY: It should be possible to specify the number of C characteristics each characteristics depend upon

   c. OPTIONAL: It should be possible to decide whether C should be identical across characteristic or whether it should be chosen from a random distribution (with an average at the inputted C).

   d. It should be defined at the beginning of the simulation which C each characteristic depends upon. This could be decided:

      i. By choosing them randomly from the set of characteristics of each species

      ii. By taking the first C characteristics in the characteristic string of each species

C.2 Traversing the landscape

1. Species of organisations should be able to walk over a fitness landscape and each will walk over a different landscape. A walk begins at a random location on the landscape, species of organisations are placed on the landscape in S-tuples (where S is the number of species).

2. It is assumed that a whole species occupies one location of its landscape. This is a simplification in the model and the affects of removing this simplification can be tested (OPTIONAL) by changing a species such that it is a cloud over the landscape (assuming agents within the organisation have similar locations).

3. At each step of the simulation every species in every S-tuple will make a move across their landscape. This move will change the current fitness of the other species in the S-tuple and the fitness available around them.

4. An species of organisation will walk over the landscape by examining its neighbours. Each location has D = (A - 1)N one mutant neighbours, a one mutant neighbour is a neighbouring location that varies from the current one by the state of one characteristic.

   a. OPTIONAL: It should be possible to specify a threshold such that if the change gives under the threshold improvement the change will not be taken:

      i. If no threshold is specified any change may be taken

ii. If a threshold is specified the change will only be taken if it gives improvement greater than the threshold (McKelvey & Yuan (2004))

b. OPTIONAL: Alternative methods of calculating fitness include:

i. Greedy Dynamics (as previously specified)

ii. Fitter Dynamics (as previously specified)

Appendix D

Detailed description of parameters

Below is an alphabetical listing of all parameters available in the simulation, where a sub-list is shown for a parameter these are the options available. The CAPITALISED options are what should be selected form the drop down list in the GUI to select this option, the number in brackets e.g. (0) is the number that must be entered to select this same option if a batch parameter file is used.

A_identical_or_random: (NK or NKC) whether the number of states given for parameter A is used as a fixed value or is used as an average value and the number of sates for each characteristic is chosen randomly with the centre of the range at A.

- IDENTICAL(0): all characteristics have the same number of states
- RANDOM(1): the number of states each characteristics is assigned is chosen at random from a uniform distribution
- GAUSSIAN(2): the number of states each characteristics is assigned is chosen at random from a normal distribution

A_size_of: (NK or NKC) an integer specifying the number of states each characteristic can be in.

C_links_between_species: (NKC) the number of characteristics from S_2 that every characteristic from species S_1 is linked to.

C_identical_or_random: (NKC) whether the number of characteristic links between species given, is used as a fixed value or is used as an average whereby C is chosen randomly for each species with the centre of the range at the given C.

- IDENTICAL(0): all characteristics are affected by the same number of characteristics from each X species they are connected to
- RANDOM(1): the number of characteristics that affect each characteristics is chosen at random from a uniform distribution
- GAUSSIAN(2): the number of characteristics that affect each characteristics is chosen at random from a normal distribution

C_size_of: (NK or NKC) an integer specifying the number of links each characteristic has with characteristics of each other species X, that is connected to.

collect_data: (NK and NKC) whether the data is collected to screen, to file or both

- ON SCREEN(0): graphs are shown on screen (this makes the program run significantly slower and should only be used for development)
- TO FILE(1): data is collected to file
- BOTH(2): data is collected both on screen and to file

comms_network: (NK) a Boolean value to state whether or not communications networks are used in this simulation.

comms_network_change: (NK) a Boolean value to state communications networks can change over time or not.

comms_network_change_chance: (NK) a percentage (value can only be between 0 and 100) to state the chance of a change occurring.

comms_network_change_frequency: (NK) the frequency with which network changes can take place, if set to 1, changes will occur every tick of the simulation, if set to two changes will occur on every other tick of the simulation etc.

comms_network_connection_probability_percentage: (NK) a percentage (value can only be between 0 and 100) to give the probability that two nodes will have a connection between them, used for random density networks.

comms_network_small_world_connect_radius: (NK) if the network is a small world network what radius of nodes should the network connect. A small world network can be represented by visualising the nodes in a circle, if this parameters is set to 1 then one nodes either side of the current node will be connected to it, if set to two, 2 nodes either side of the current node will be connected to it etc.

comms_network_type: (NK) the type of network to be used, this can be a fully connected network, a linear network, a random density network or a small world network.

- LINEAR NETWORK(0): the network used is a linear network
- FULLY CONNECTED NETWORK(1): the network used is a fully connected network
- RANDOM NETWORK(2): the network used is a random density network, if this is used a comms_network_connection_probability_percentage must be set
- SMALL WORLD NETWORK(3): the network used is a small world network, if this is used a comms_network_small_world_connect_radius must be set

data_collection_file_name: (NK and NKC) the name of the file to collect data to.

fitness_method: (NK and NKC) the method used to calculate the fitness of an organisation, this can either be average or weakest.

- AVERAGE(0): the fitness of an organisation is taken as an average over all characteristics, if this is set fitness_method_average_weightings must also be set
- WEAKEST(1): the fitness of an organisation is taken as the fitness of the weakest characteristic

fitness_method_average_weightings: (NK and NKC) if the fitness_method is set to average, weightings can either be turned on or off.

- IDENTICAL(0): the fitness of an organisation is taken as an identical average

- RANDOM(1): the fitness of an organisation is taken as a weighted average

fitness_range_dp: (NK and NKC) an integer to specify the number of decimal places the fitness range covers i.e. if set to 1 the range would be 0.0 to 1.0, however if set to 2 the range would be 0.00 to 1.00 (default is 2 decimal places).

fitness_threshold: (NK and NKC) a double, if this is set to 0.0 then it is not used, otherwise if the difference in fitness between one location and the next is not greater than the given threshold then the organisation will not move to the new location.

jump_J: (NK) this determines whether the organisation is capable of jumping across the landscape.

- WALK(0): the organisation may only walk across the landscape
- LONG JUMP(1): the organisation may both walk across the landscape and them when a peak is reached it may jump to find a fitter location

jump_search_limit: (NK) this integer puts a limit on how long the organisation can search for a new jump for (if set to zero this parameter is not used).

jump_successful_limit: (NK) this integer puts a limit on the number of successful jumps an organisation is allowed to take (if set to zero this parameter is not used).

K_identical_or_random: (NK and NKC) whether K, for every characteristic, is the K given or whether it is random with an average of K.

- IDENTICAL(0): all characteristics are affected by the same number of other characteristics from within the same organisation
- RANDOM(1): each characteristic is affected by a different number of other characteristics and this is chosen from a uniform random distribution with the average at K
- GAUSSIAN(2): each characteristic is affected by a different number of other characteristics and this is chosen from a normal random distribution with the average at K

K_neighbours_or_random: (NK and NKC) whether the K characteristics that N depends upon are taken form the neighbours of that N, or whether they are drawn at random from the set of characteristics.

- NEIGHBOURS(0): the K characteristics are chosen as neighbours of N
- RANDOM(1): the K characteristics are chosen at random form the set of all N

K_size_of: (NK and NKC) an integer specifying the number of characteristics each characteristic depends on for its fitness calculation.

life_and_death_new_org_method: (NK) if life and death is being used then the method to create a new organisation must be chosen, the initial location of this organisation can be either:

- RANDOM NEW ORG(0): chosen at random
- COPY OLD ORG(1): a copy of a current organisation
- BOTH(2): dependant on the genetic load of the landscape the initial location may be either chosen at random or copied from a current location

life_and_death_threshold: (NK) a double value such that each organisation is compared to the fittest organisation on the current landscape, if the difference in fitness is below this threshold then the organisation dies.

N_size_of: (NK and NKC) an integer specifying the number of characteristic of each organisation (and therefore of each location on the landscape).

next_neighbour_method: (NK and NKC) the method by which the next neighbouring location is chosen to examine. Having chosen the neighbour it will then be checked to see if it is fitter than the current location. (NB: this is not relevant for Fitter Dynamics).

- SYSTEMATIC(0): the nearest neighbours of the location are looked at in order
- RANDOM WITH MEM(1): the nearest neighbours of the location are looked at, at random but without looking at the same one twice
- RANDOM NO MEM(2): the nearest neighbours of the location are looked at, at random with no memory or what has already been looked at

organisations_no_of: (NK and NKC) an integer specifying the number of organisation that are thrown onto the landscape at the beginning of the simulation (default is 100)

organisational_walk_type: (NK and NKC) the organisation could use either:

- ONE MUTANT NEIGHBOUR(0): where one neighbouring location is looked at on each tick
- FITTER DYNAMICS(1): where on each tick neighbouring locations are looked at in turn until a fitter one is found
- GREEDY DYNAMICS(2): where all neighbouring locations are looked at on each tick and the fittest one is chosen

S_species: (NKC) an integer specifying the number of species of organisation in the simulation.

simulation_halt: (NK and NKC) an integer specifying the tick at which the simulation should stop (this has no effect if set to zero).

X_species: (NKC) an integer specifying how many other species each species of organisation is connected to.

X_neighbours_or_random: (NKC) whether the X species are taken as the neighbouring species of the species S1, or whether they are chosen at random form the set of all species.

- NEIGHBOURS(0): the species each species is connected to are selected as the neighbours of that species
- RANDOM(1): the X species each species is connected to are selected at random from the set of all species

X_identical_or_random: (NKC) whether X is set to the given X for all species or whether X is chosen randomly for each species, with an average of the given X.

- IDENTICAL(0): the number X species each species is connected to is identical for all species
- RANDOM(1): the number of X species each species is connected to is chosen from a uniform random distribution with an average of the given X
- GAUSSIAN(2): the number of X species each species is connected to is chosen from a normal random distribution with an average of the given X

xml_file: (NKC) the file name of the xml file that specifies the details of each species of organisation for this simulation.

Appendix E

Further NK Model results

The following section shows all results found during NK simulation runs that were not shown in the NK Model results section. These results are taken from different environments and they back up the results already shown.

This appendix is split into two sections, the first section gives further results for the docking of the NK Model to Kauffman's original Model and the second section gives the remaining results for the further research done.

E.1 Docking to Kauffman's model

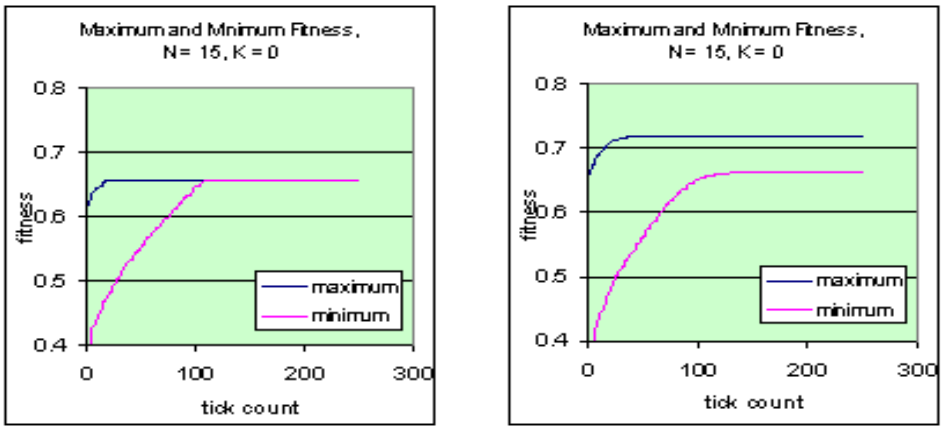

Figure E.1: The two images show the maximum, minimum and average fitness for an N = 15 simulation at both K = 0 and K = 1.

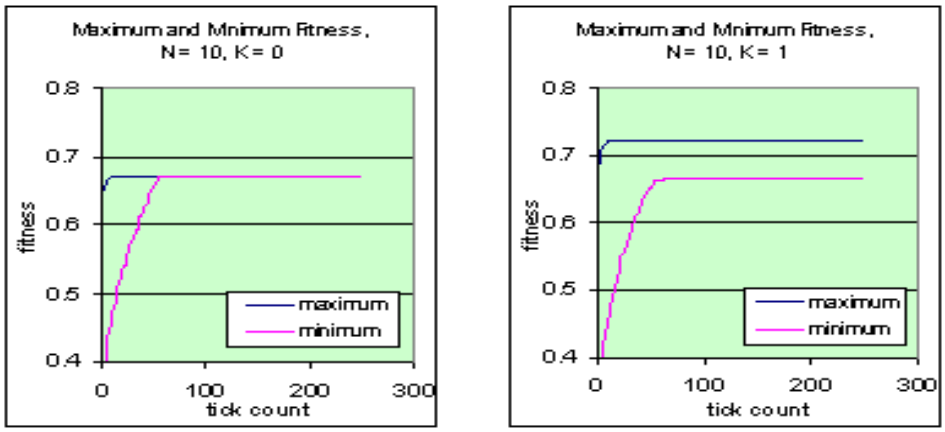

Figure E.2: The two images show the maximum, minimum and average fitness for an N = 10 simulation at both K = 0 and K = 1.

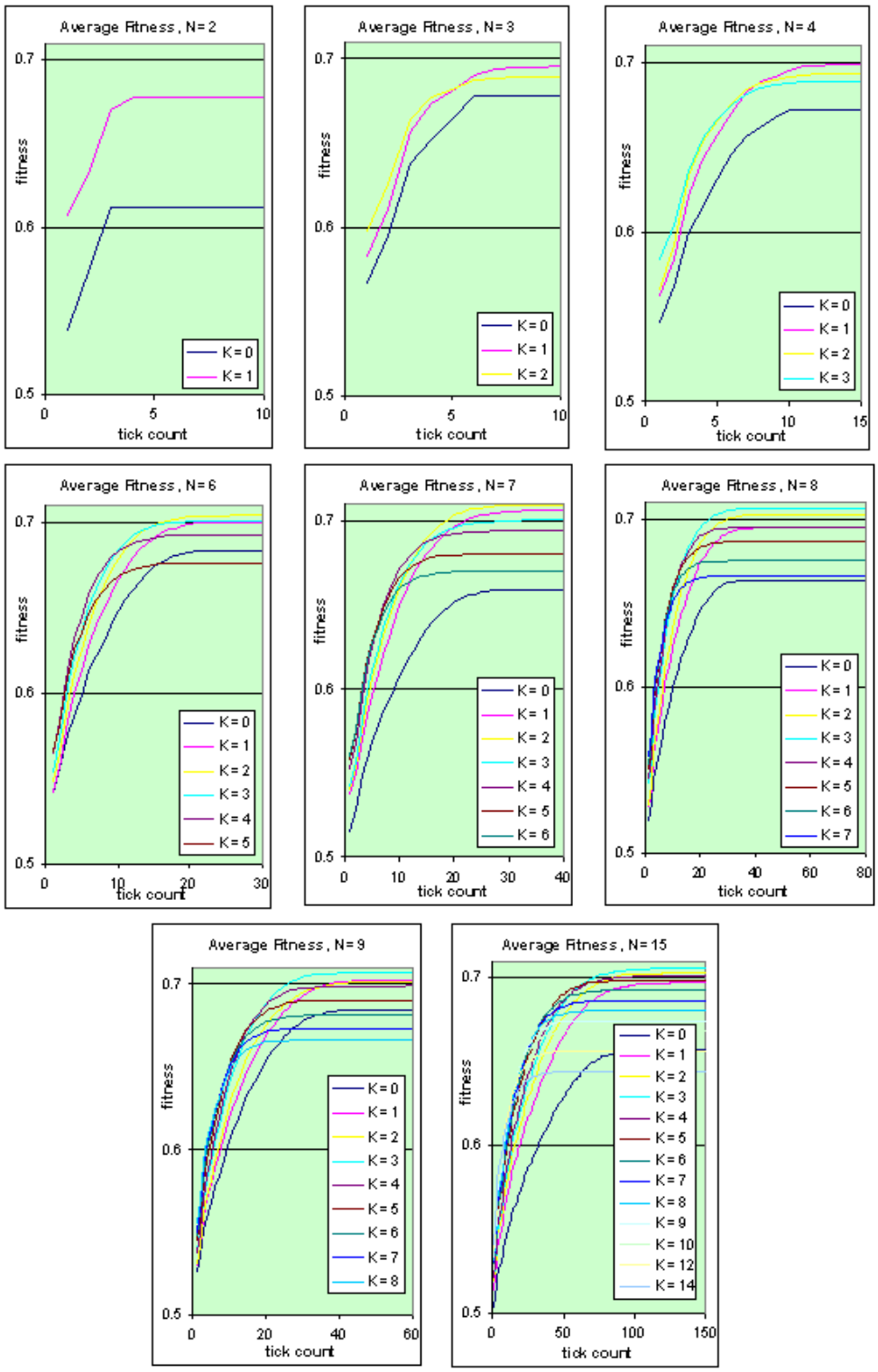

Figure E.3: Average organisational fitness at different values of N and K.

E.2 Research using the NK Model

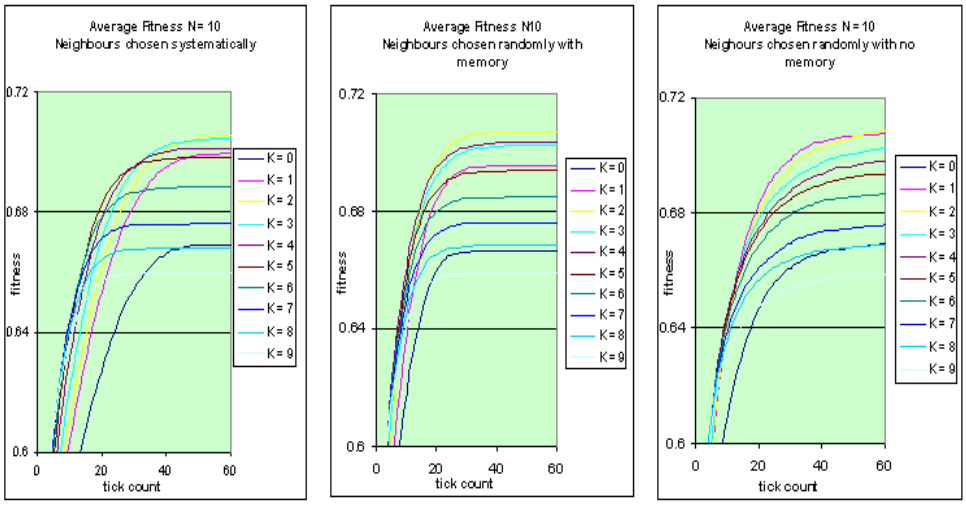

Figure E.4: The three images show K increasing for N = 10, using the three different methods of choosing the next neighbour.

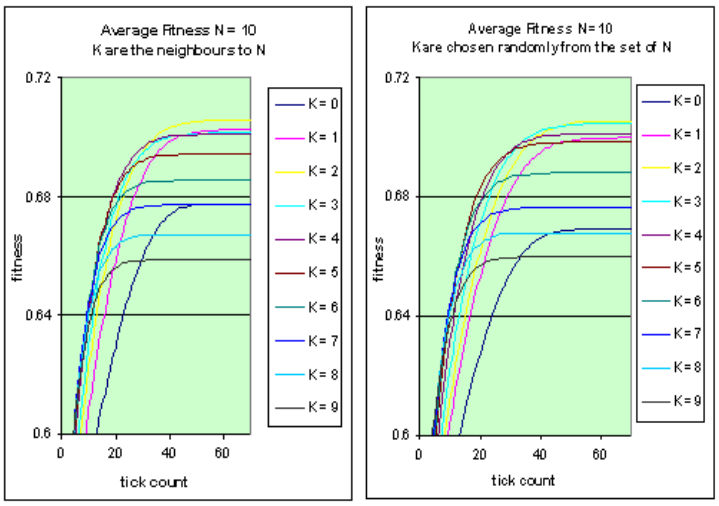

Figure E.5: The two images show K increasing for N = 10, in the left image K are chosen as the neighbours to N, in the right image K are chosen randomly.

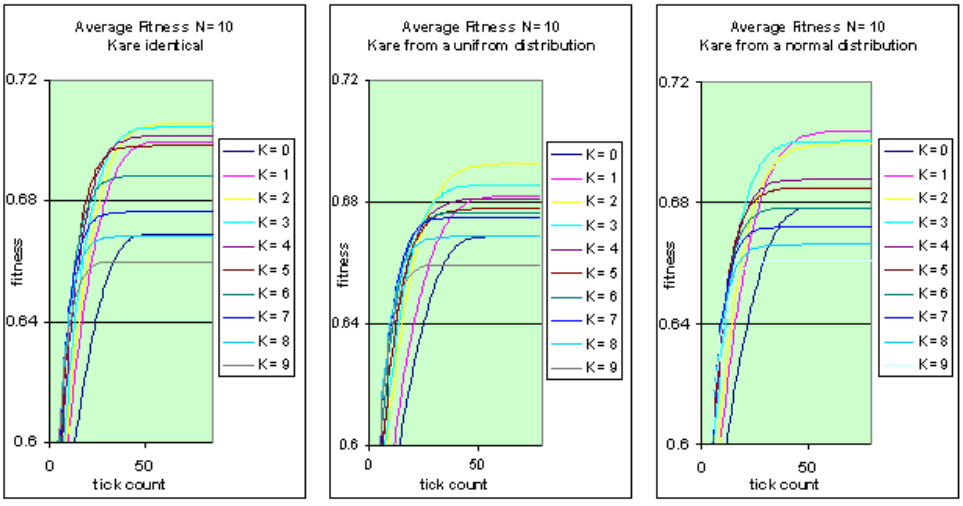

Figure E.6: The three images show K increasing for N = 10; the graphs show where K is identical, chosen from a uniform random distribution and chosen from a normal random distribution respectively.

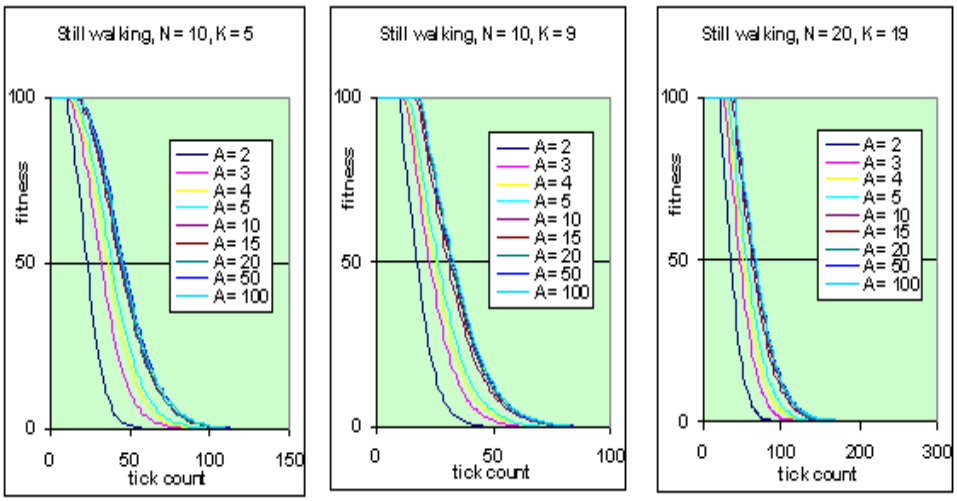

Figure E.7: The above images show the number of organisations still walking at each tick whilst A is varied.

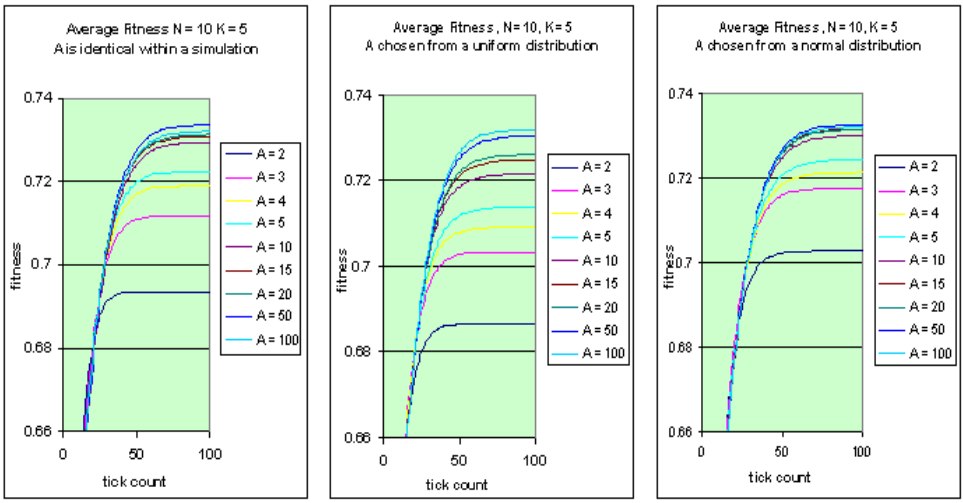

Figure E.8: The three images show A increasing for N = 10 and K = 5; the graphs show where A is identical, chosen from a uniform random distribution and chosen from a normal random distribution respectively.

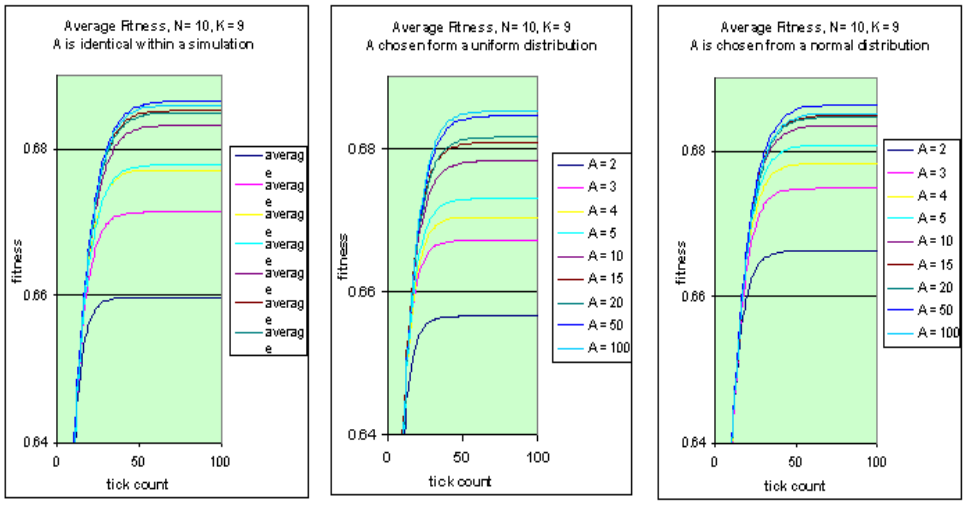

Figure E.9: The three images show A increasing for N = 10 and K = 9; the graphs show where A is identical, chosen from a uniform random distribution and chosen from a normal random distribution respectively.

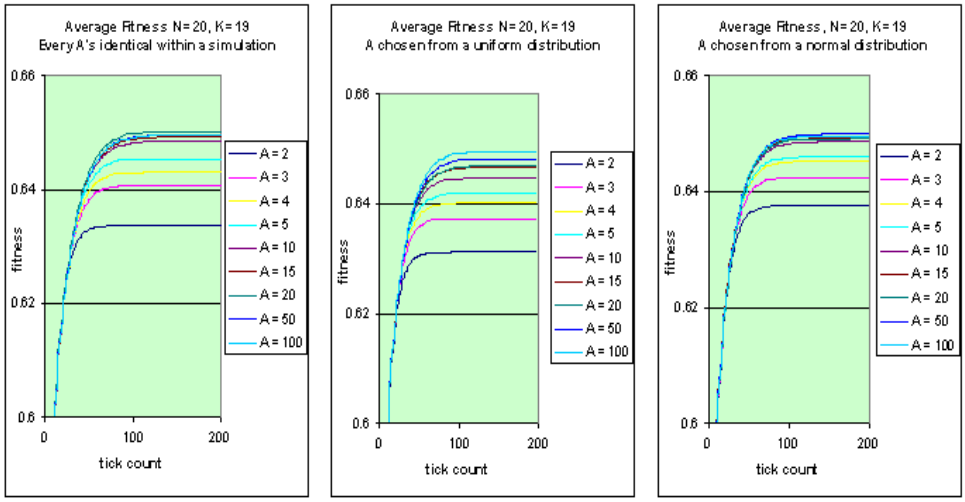

Figure E.10: The three images show A increasing for N = 20 and K = 19; the graphs show where A is identical, chosen from a uniform random distribution and chosen from a normal random distribution respectively.

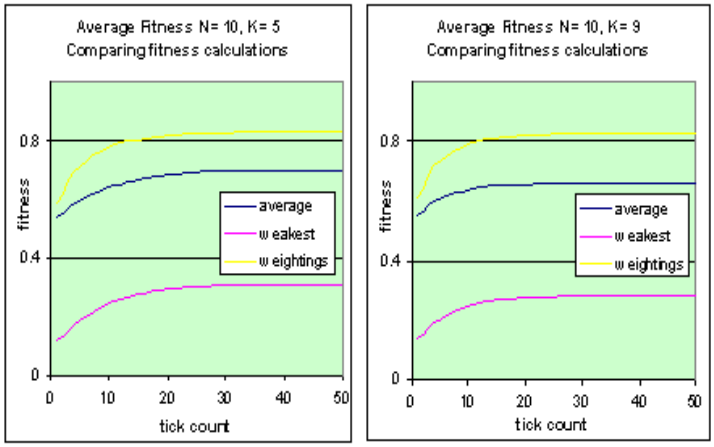

Figure E.11: The two images show the three different methods of calculating fitness for N = 10 at two different values of K.

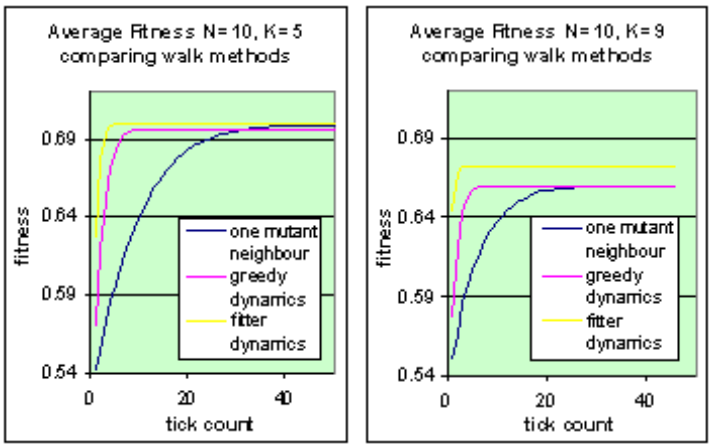

Figure E.12: The two images show three different methods of walking across the landscape for N = 20 and two different values of K.

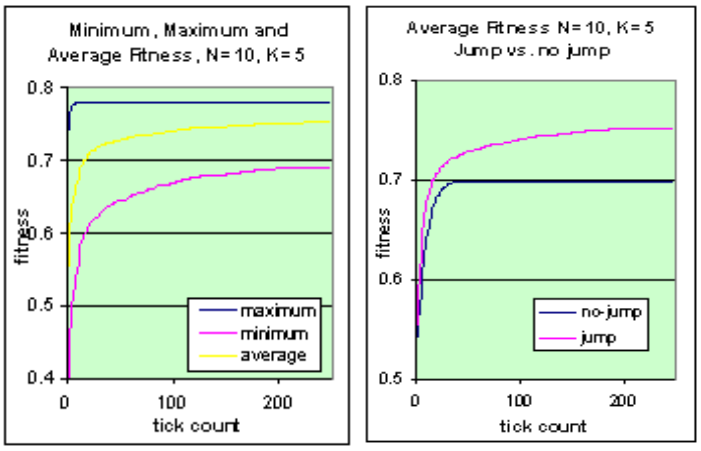

Figure E.13: The graph to the left shows the maximum, minimum and average fitness of 100 organisation as they move and jump across the landscape for N = 10 and K = 5. The graph to the right shows the difference between the average fitness of a set of organisations that can jump across the landscape and a set of organisations that cannot.

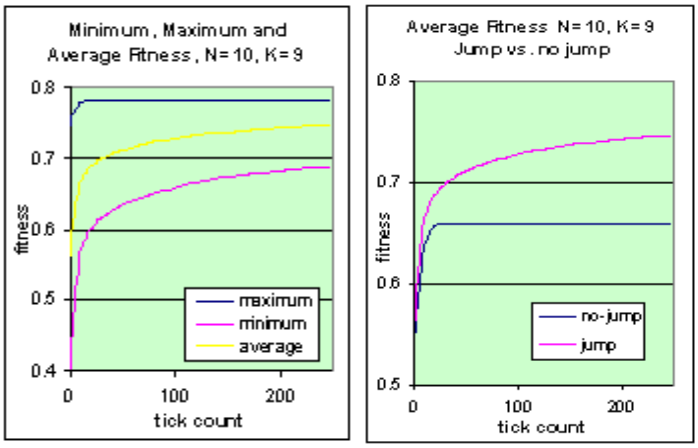

Figure E.14: The graph to the left shows the maximum, minimum and average fitness of 100 organisation as they move and jump across the landscape for N = 10 and K = 9. The graph to the right however shows the difference between the average fitness of a set of organisations that can jump across the landscape and a set of organisations that cannot jump.

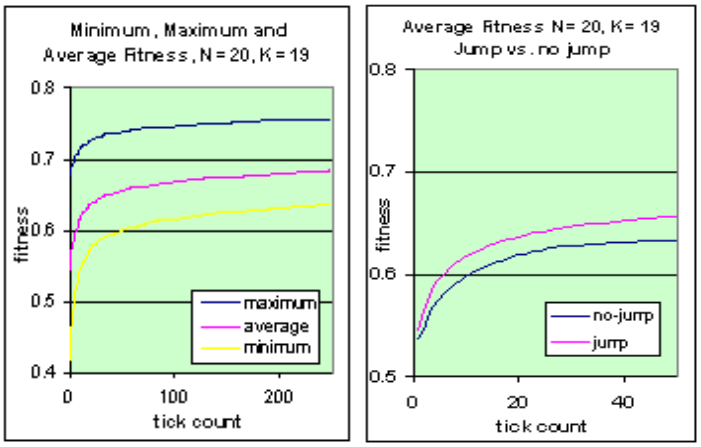

Figure E.15: The graph to the left shows the maximum, minimum and average fitness of 100 organisation as they move and jump across the landscape for N = 20 and K = 19. The graph to the right however shows the difference between the average fitness of a set of organisations that can jump across the landscape and a set of organisations that cannot jump.

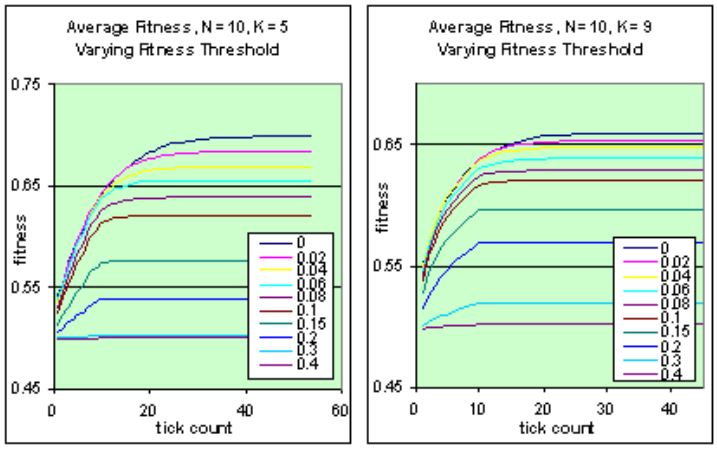

Figure E.16: The two graphs show changing fitness threshold for N = 10 at two different values of K.

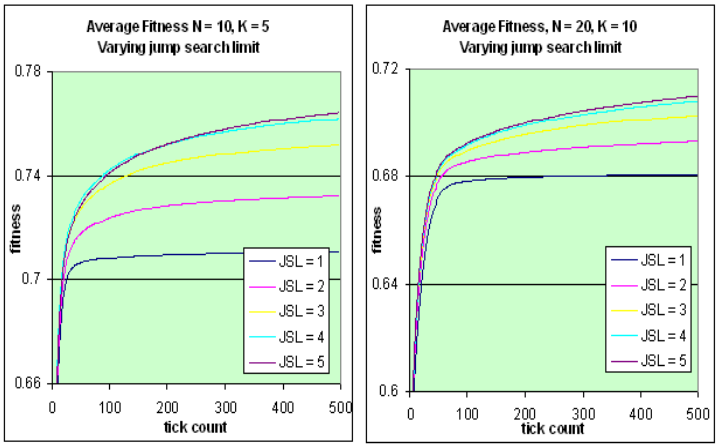

Figure E.17: The two graphs show how changing the number of jumps allowed affects the fitness gained.

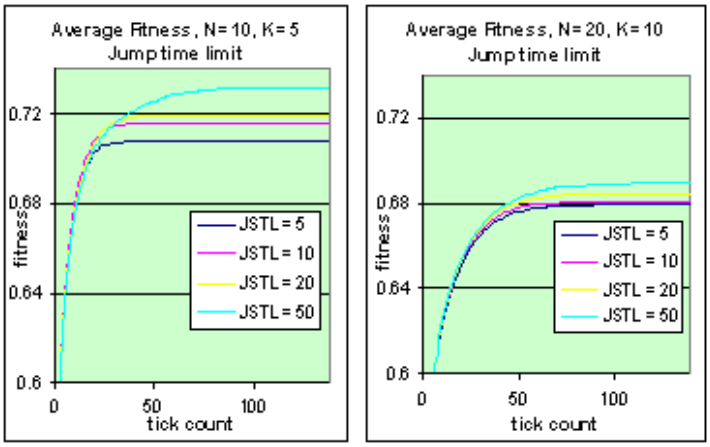

Figure E.18: The two graphs show how changing the length of time an organisation is allowed to search for a new jump for affects the fitness gained.

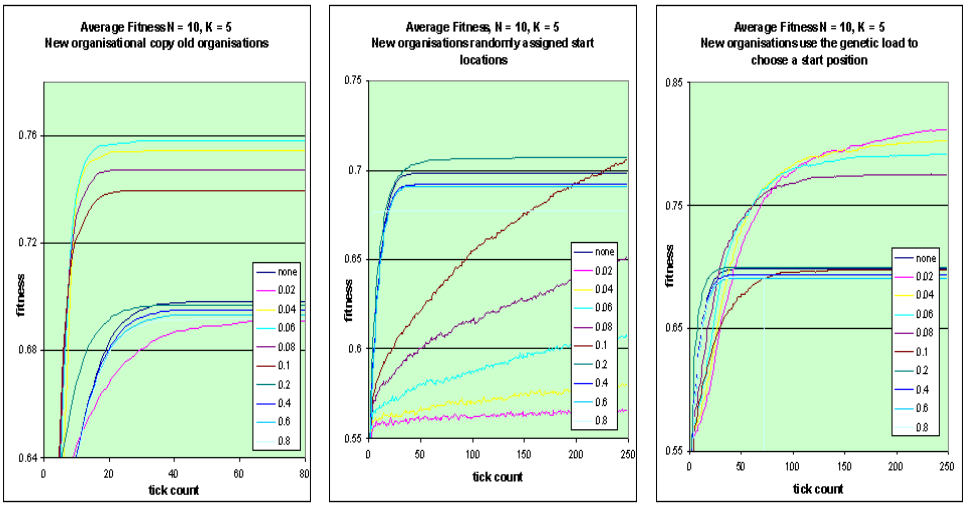

Figure E.19: The three images show a varying death_threshold for the three different methods of new organisation creation.

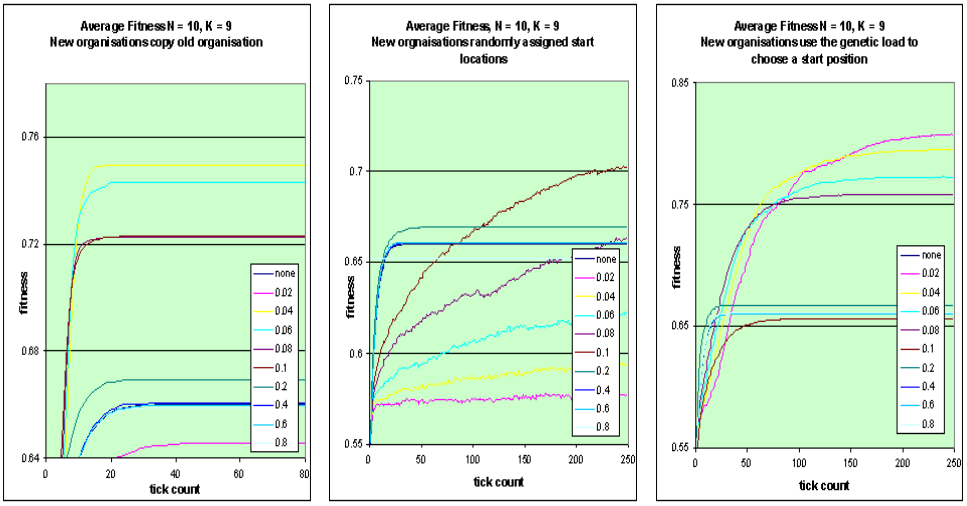

Figure E.20: The three images show a varying death_threshold for the three different methods of new organisation creation.

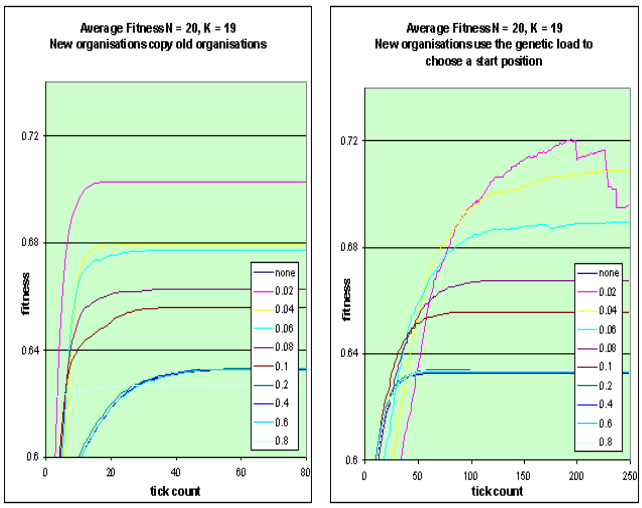

Figure E.21: The two images show a varying death_threshold for the two different methods of new organisation creation.

Appendix F

Further NKC Model results

The following section shows all results found during NKC simulation runs that were not shown in the NKC Model results section. These results are taken from different environments and they back up the results already shown.

This appendix is split into two sections, the first section gives further results for the docking of the NKC Model to Kauffman's original Model and the second section gives the remaining results for the further research done.

F.1 Docking the model to Kauffman's model

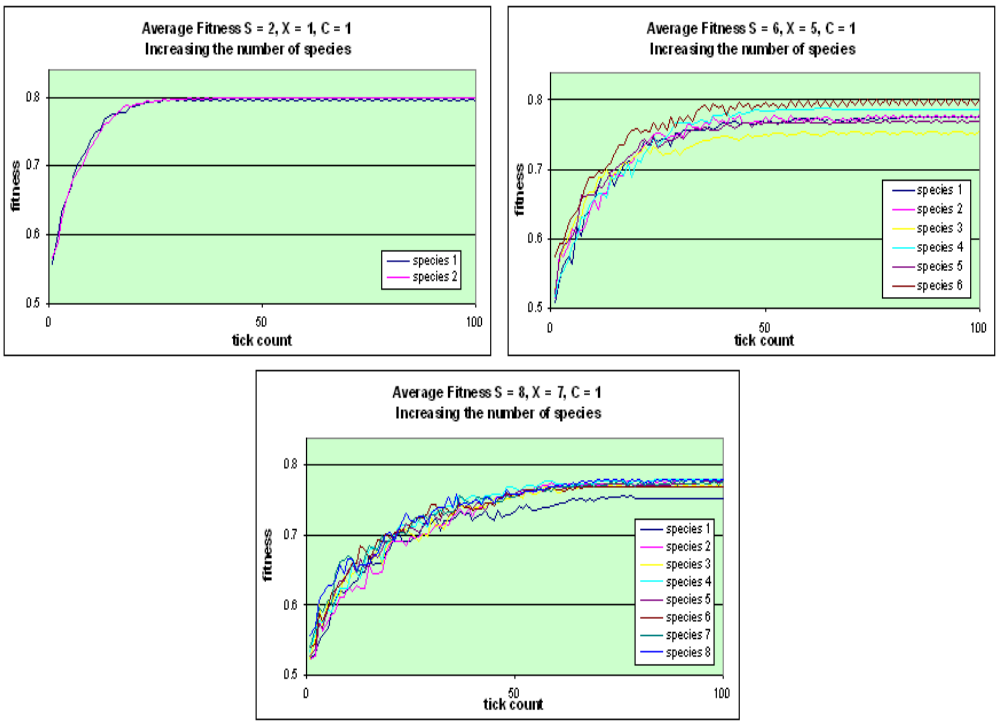

Figure F.1: The above images show sets of fully connected, co-evolving species. The results show differing numbers of co-evolving species, shown S = 2, S = 6 and S = 8.

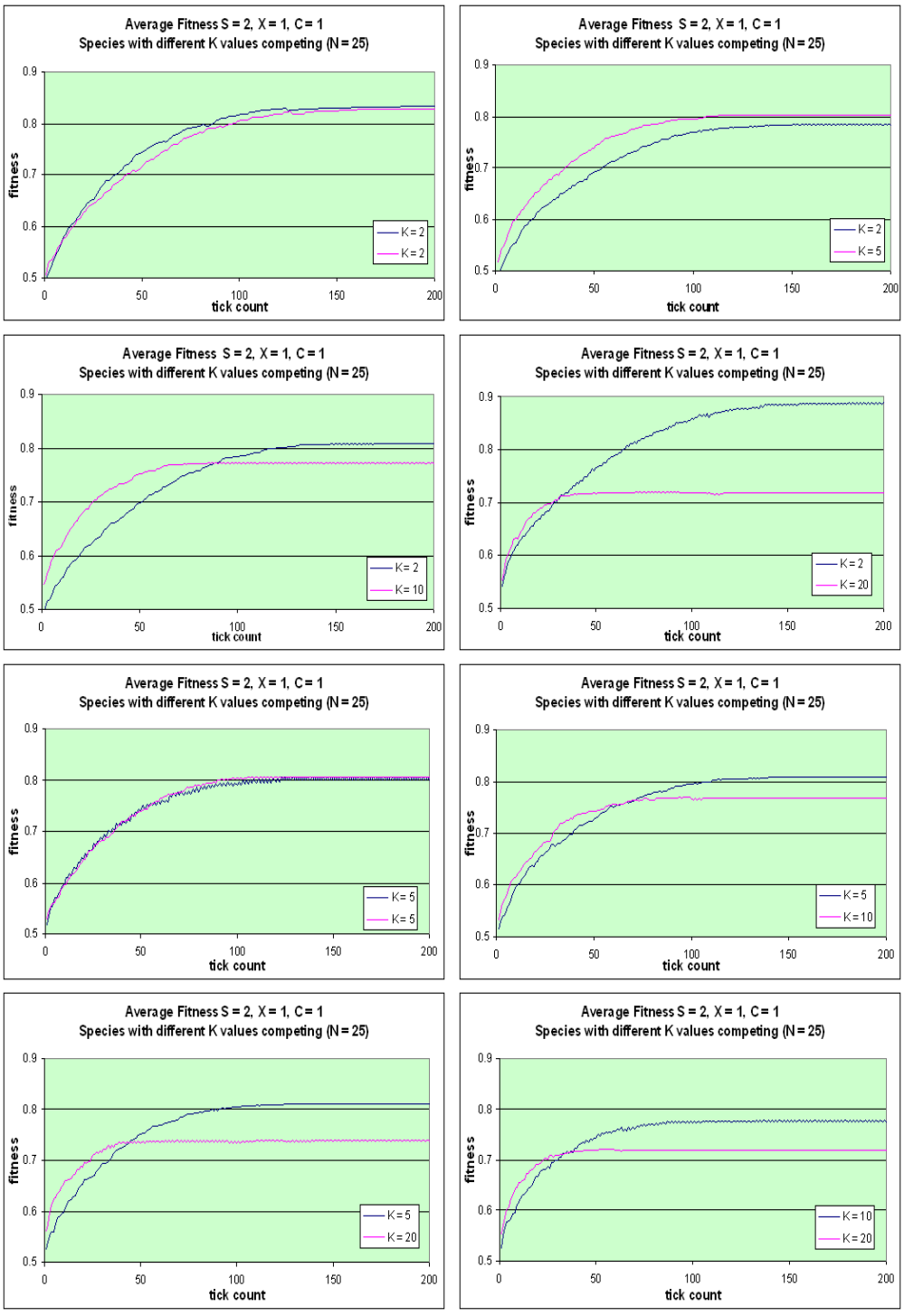

Figure F.2: The above graphs show co-evolutionary pairs of species competing for higher fitness at C = 1, the affects of varying K within these competing species are investigated.

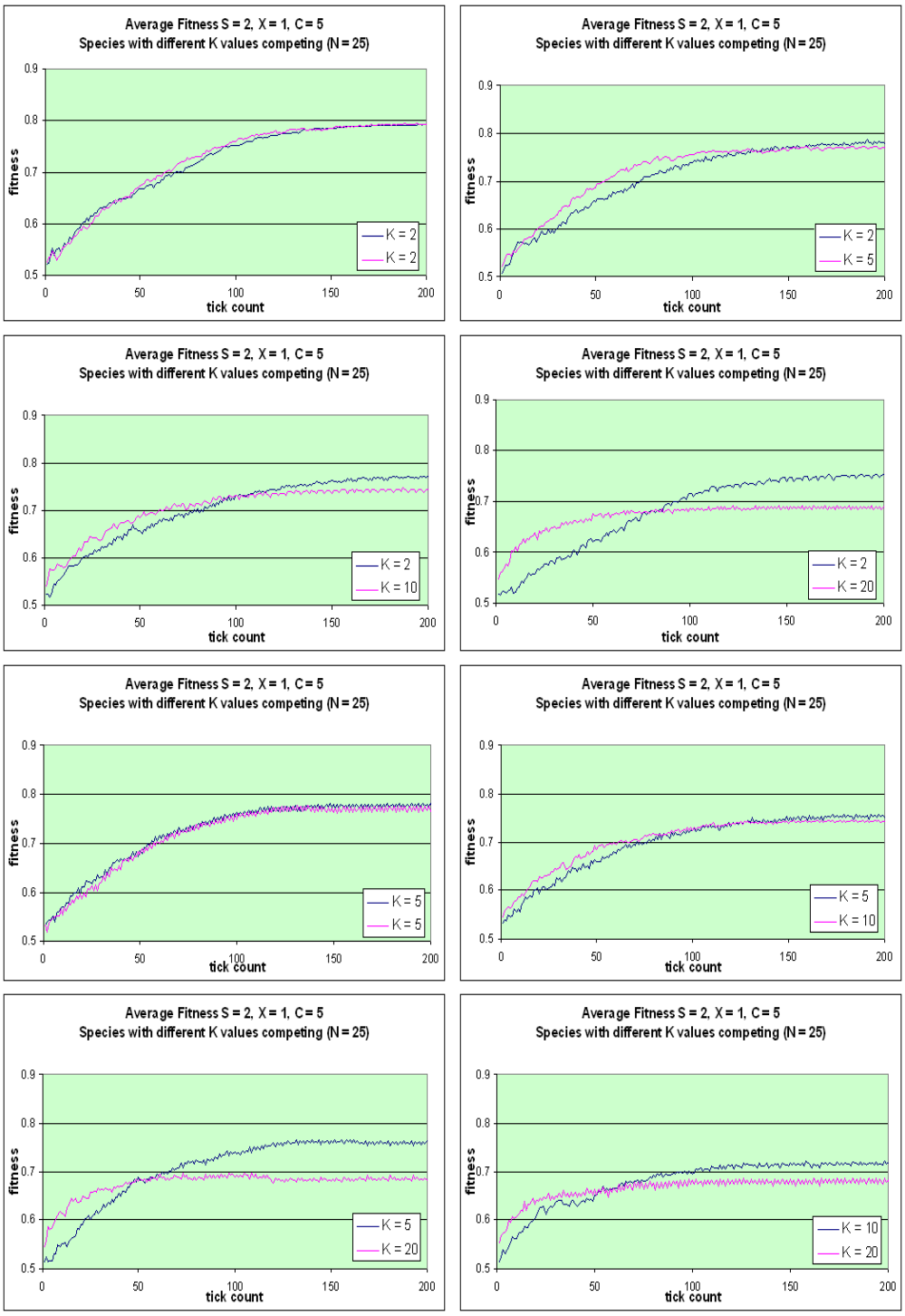

Figure F.3: The above graphs show co-evolutionary pairs of species competing for higher fitness at C = 5, the affects of varying K within these competing species are investigated.

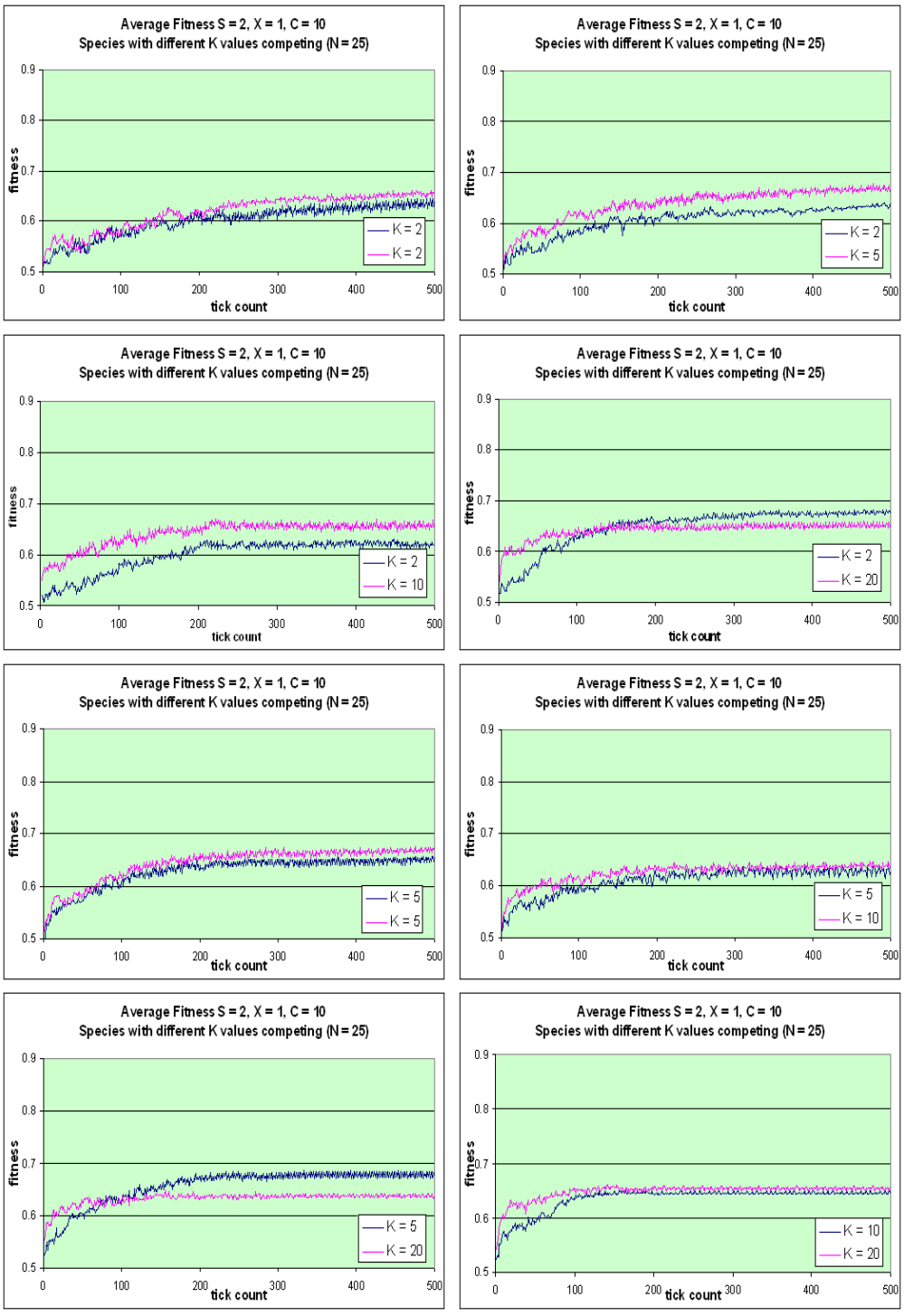

Figure F.4: The above graphs show co-evolutionary pairs of species competing for higher fitness at C = 1, the affects of varying K within these competing species are investigated.

F.2 Research using the NKC Model

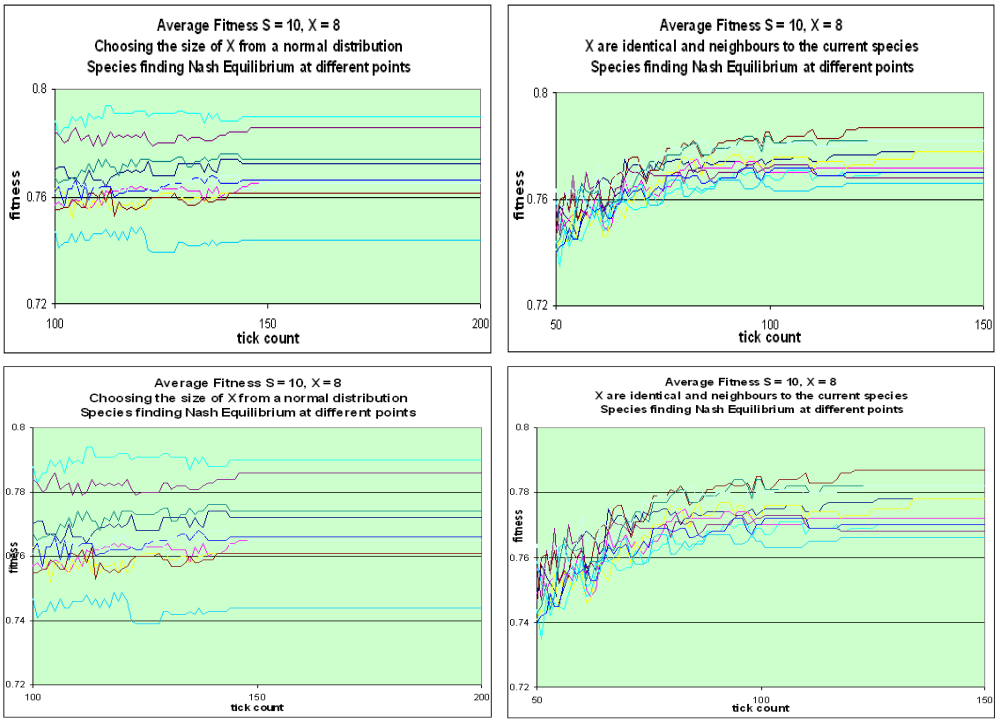

Figure F.5: This figure shows an example from all simulations done whilst changing X, it show a close up version of Nash Equilibrium being reached at different points by different species.

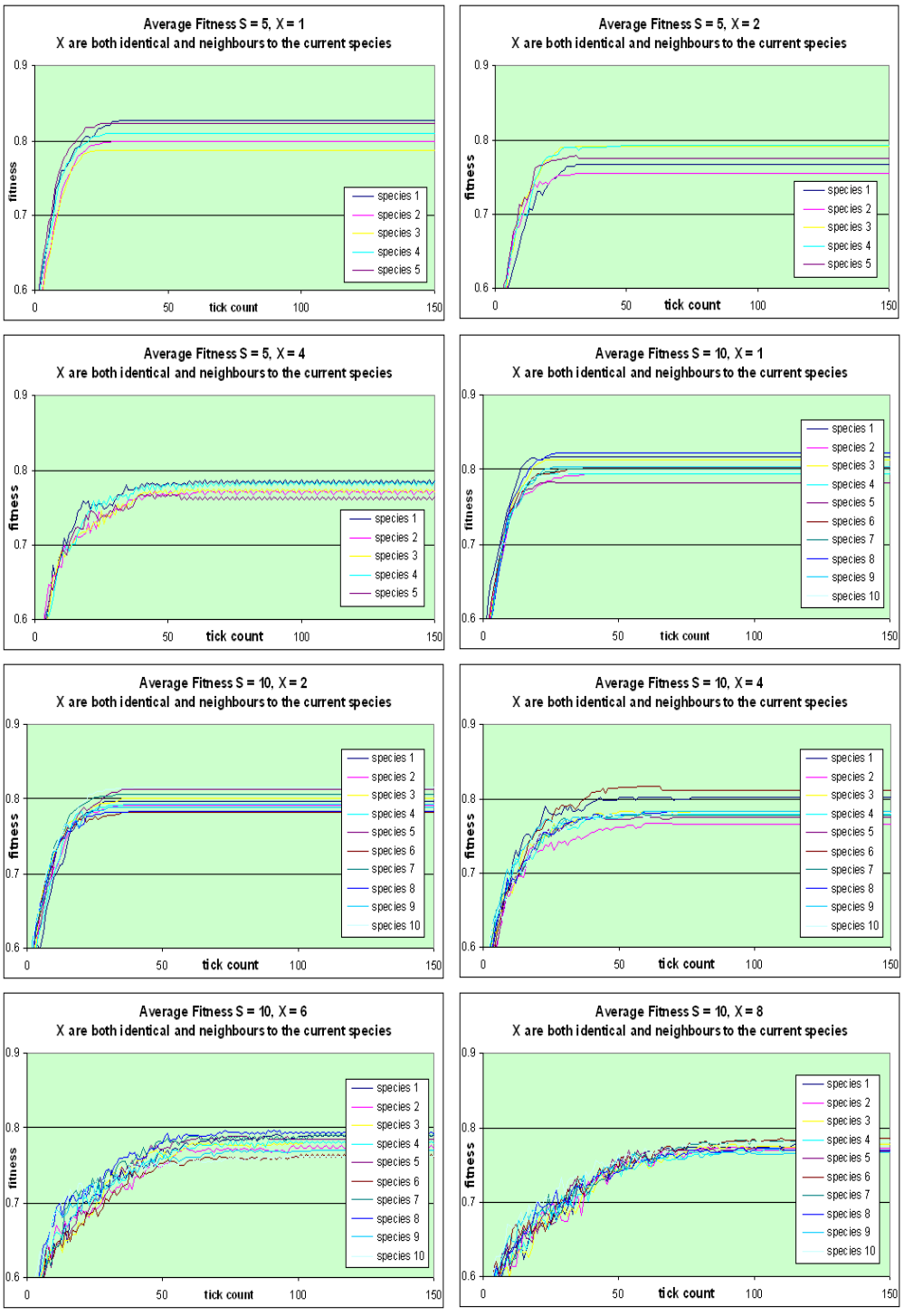

Figure F.6: This figure shows an example from all simulations done whilst changing X, with X set to identical between species.

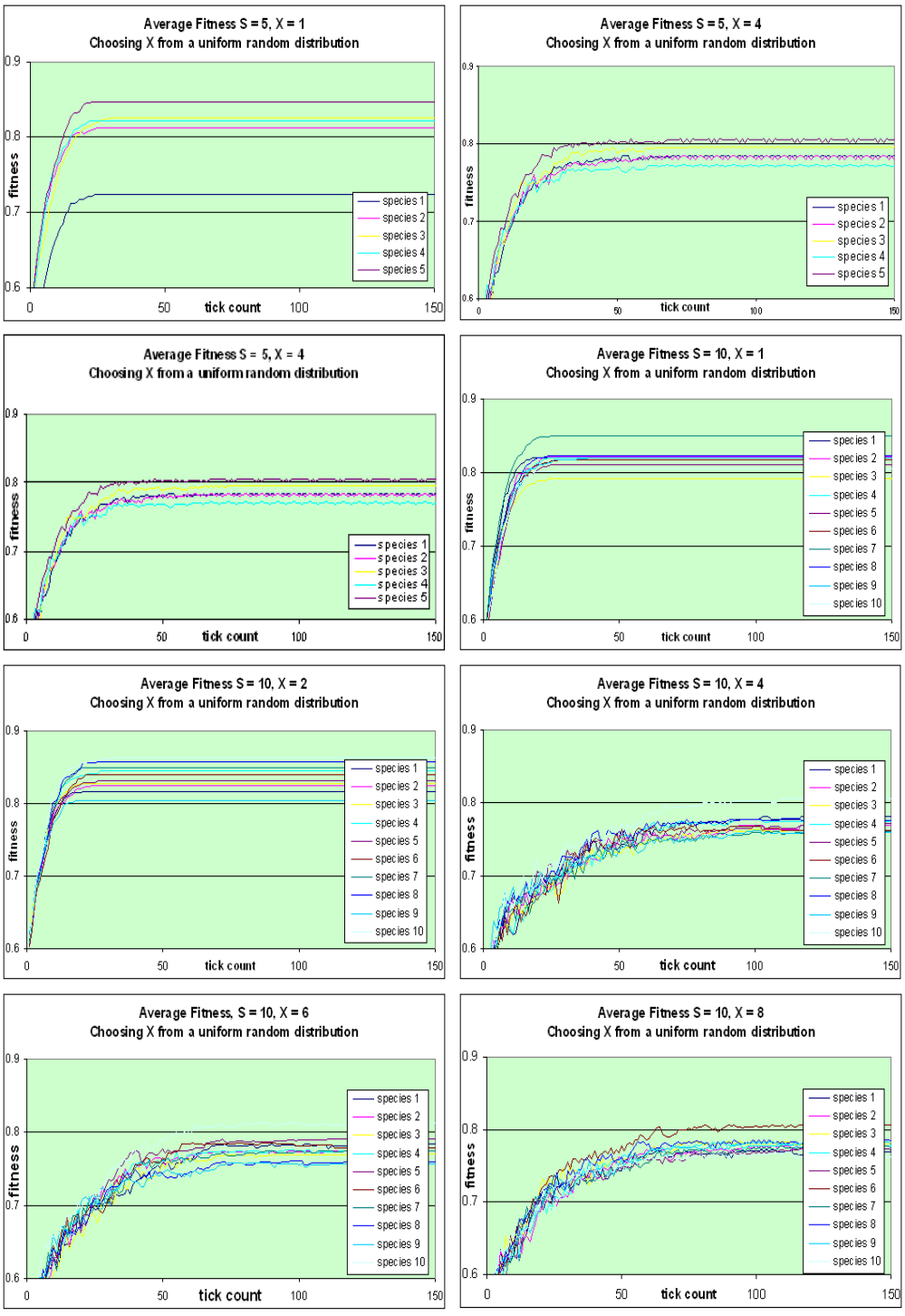

Figure F.7: This figure shows an example from all simulations done whilst changing X, with the size of X set to be taken from a uniform random distribution.

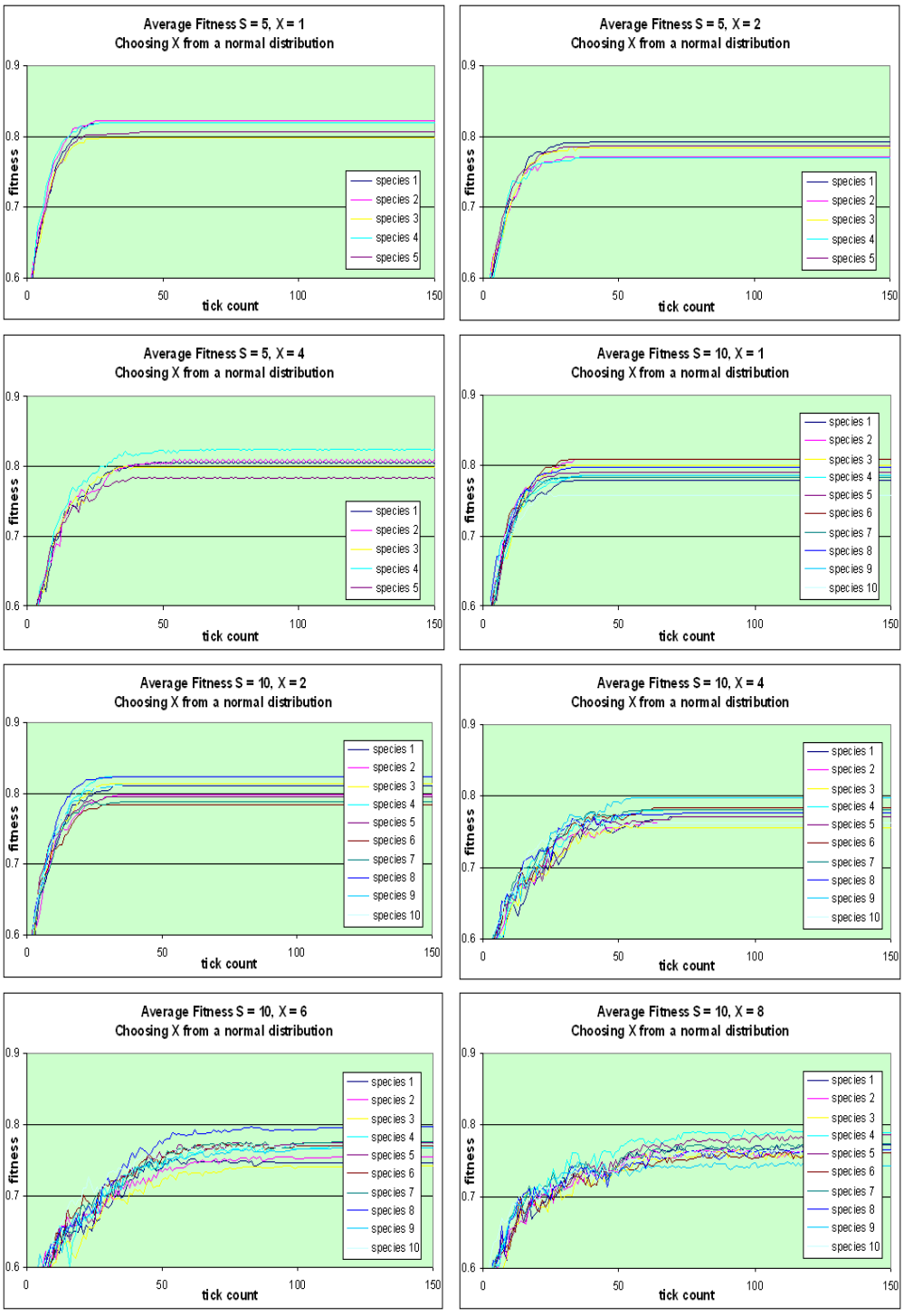

Figure F.8: This figure shows an example from all simulations done whilst changing X, with the size of X set to be taken from a normal random distribution.

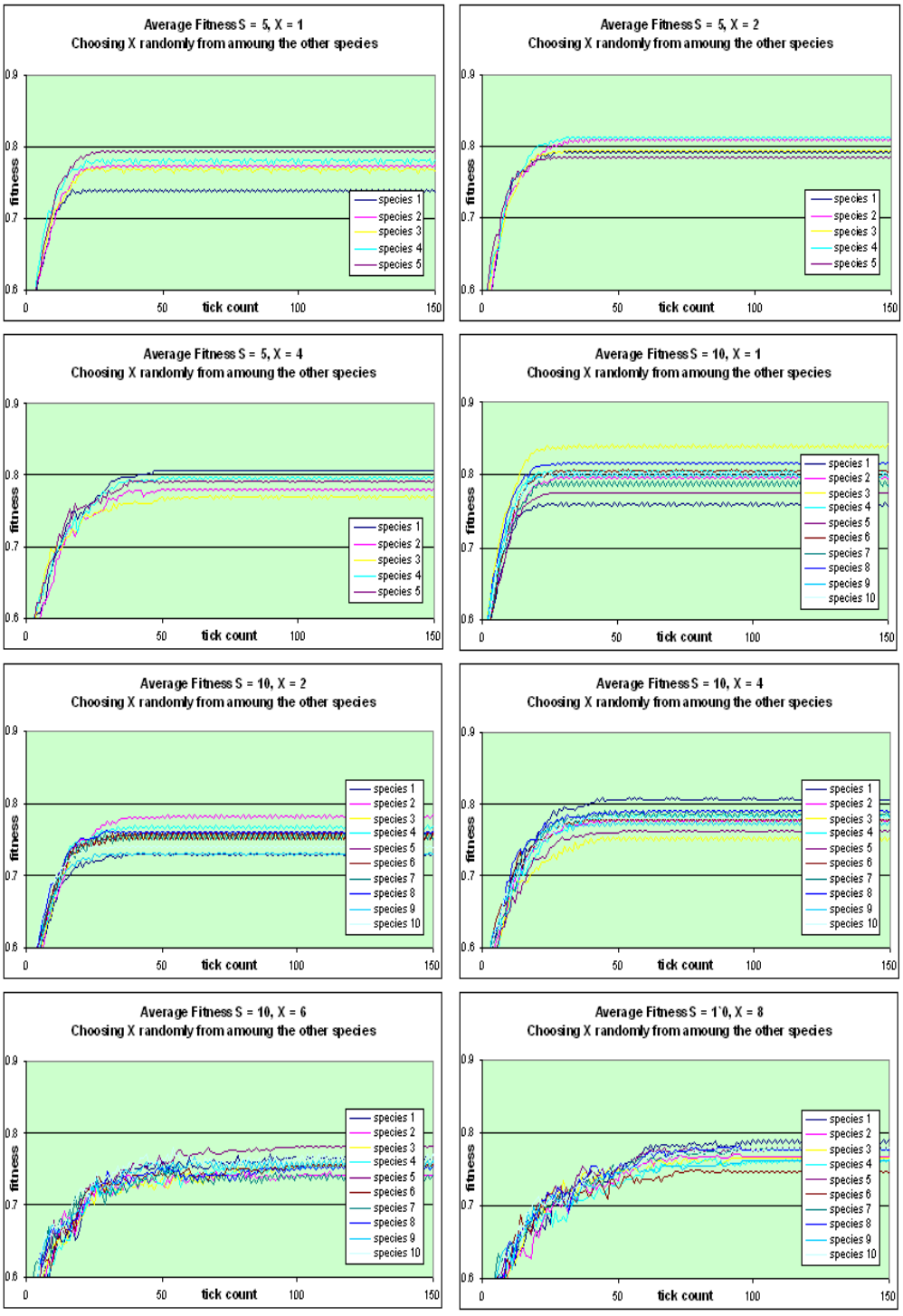

Figure F.9: This figure shows an example from all simulations done whilst changing X, where X are chosen randomly from the set of all species (rather than being taken as the neighbours of the current species).

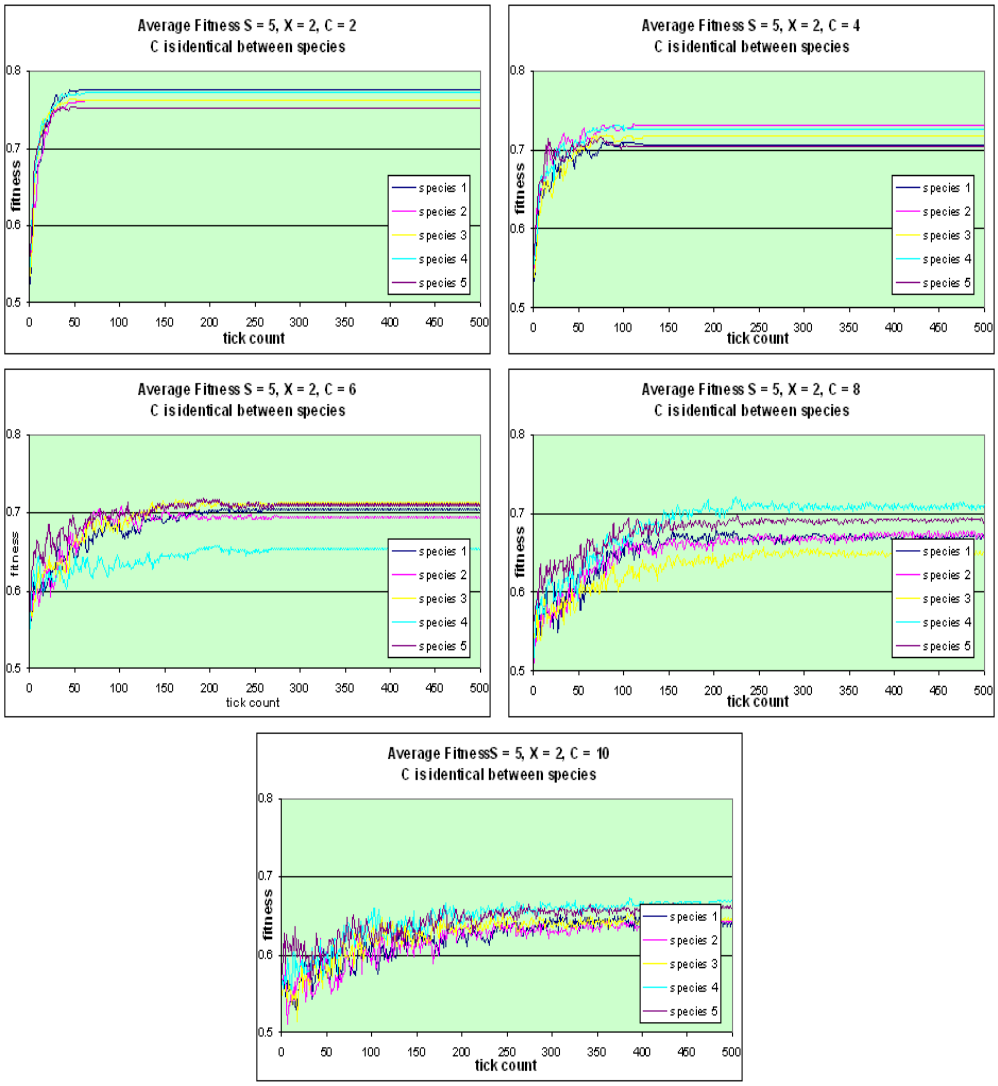

Figure F.10: This figure shows an example from all simulations done whilst changing C where the size of C is identical between species.

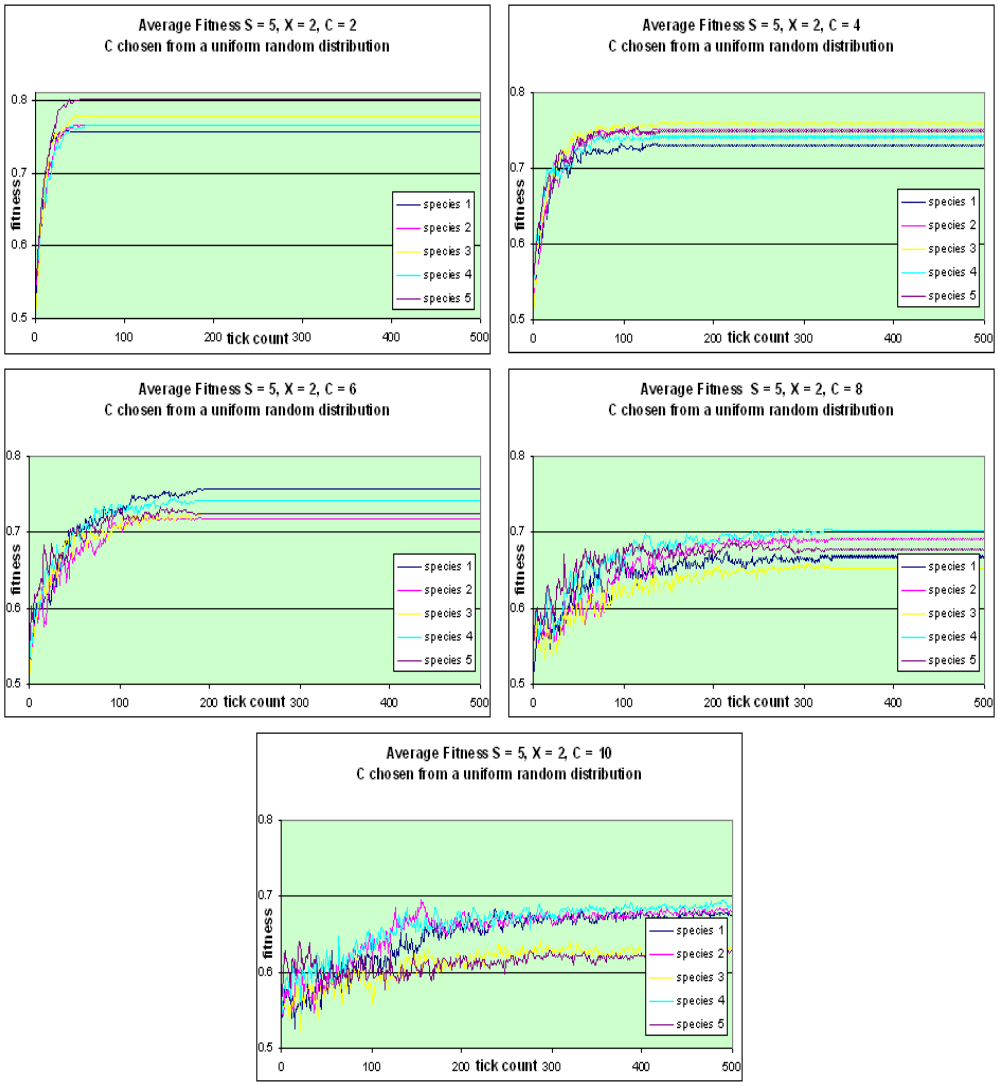

Figure F.11: This figure shows an example from all simulations done whilst changing C where the size of C is taken from a uniform random distribution with an average at the inputted C.

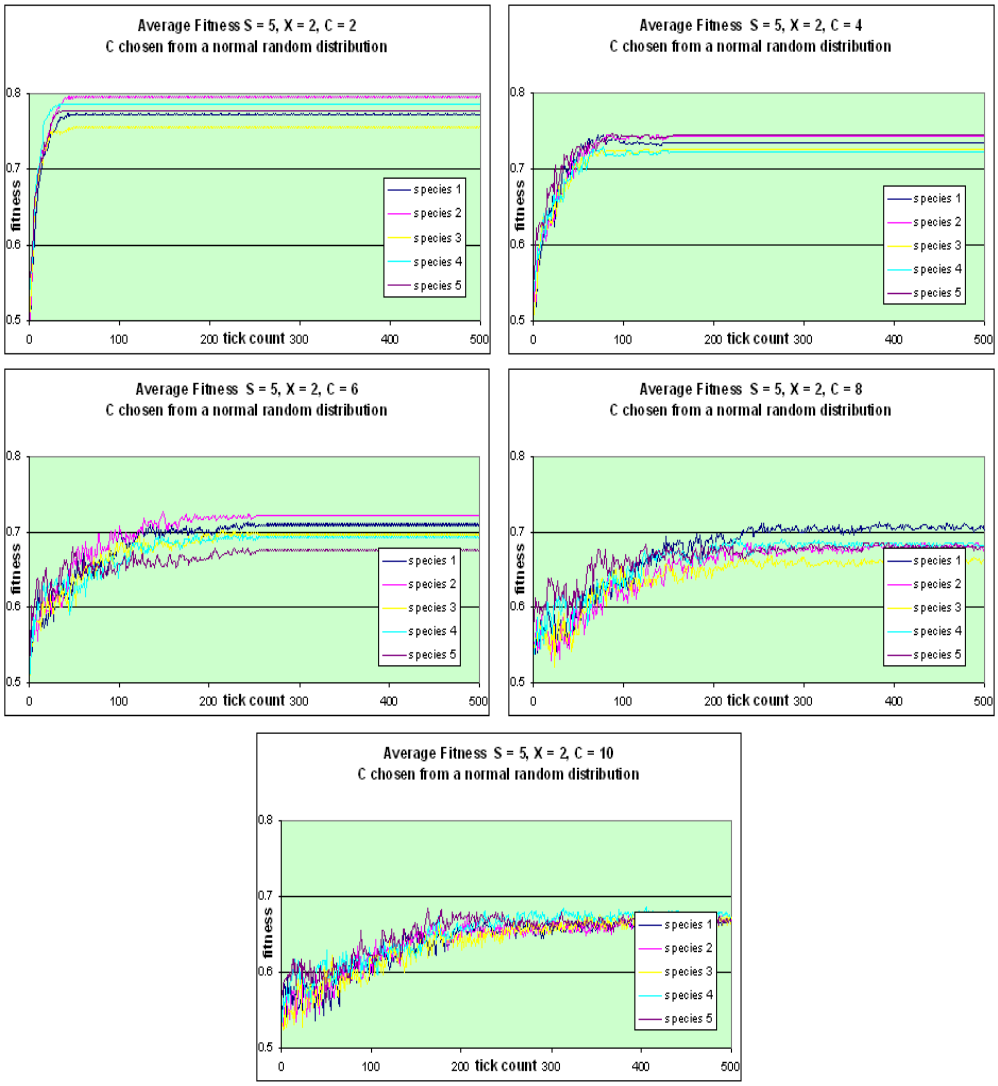

Figure F.12: This figure shows an example from all simulations done whilst changing C where the size of C is taken from a normal random distribution with the average at the inputted C.

Appendix G

Code

The following section contains a subset of the code from this project, all code can be found in on the attached CD. The code is comprised of four packages:

- The CrossModelClasses package contains two abstract classes and a parameter class that can be used by both models. This includes the following classes:
  - ParameterOptions: the class where static final values are stored, to be used by all other classes
  - AbstractFitnessLandscape: a specification of the basic fitness landscape
  - AbstractOrganisation: a specification of the basic organisation
- The DataCollection package contains all classes to facilitate data collection. This includes the following classes:
  - Fitness: the class that calculates the maximum, minimum and average fitness, over all organisations in a simulation
  - FractionStillWalking: the class that calculates the fraction of organisations that are still walking
  - NKCDataCollector: the class that coordinates the data collection for the NKC model
  - NKDataCollector: the class that coordinates the data collection for the NK Model
  - NoFitterNeighbours: the class that calculates the maximum, minimum and average number of fitter neighbours, over all organisation in a simulation
  - WaitTime: the class that calculates the maximum, minimum and average time an organisations had to wait before it jumps, over all organisations in a simulation
- The NKModel package contains all classes relating to the NK Model. This includes the following classes:
  - FitnessMessage: this class facilitates organisational communication by providing a message format (communication was not tested in this dissertation, for the reasons discussed)
  - NKFitnessLandscape: this class specifies the NK fitness landscape
  - NKModel: this class coordinates the working of the model
  - NKModelBatch: this class contains the main method for running the model in batch mode

  - NKModelGUI: this class contains the main method for running the model in GUI mode
  - NKOrganization: this class specifies the NK agent, the organisation
  - OrganizationEdge: this class facilitates organisational communication by providing a communication channel between organisations (communication was not tested in this dissertation, for the reasons discussed)

- The NKCModel package contains all classes relating to the NKC Model. This includes the following classes:
  - CoevolutionarySet: this class specifies the NKC agent, the co-evolutionary set
  - NKCFitnessLandscape: this class specifies the NKC fitness landscape
  - NKCModel: this class coordinates the working of the model
  - NKCModelBatch: this class contains the main method for running the model in batch mode
  - NKCModelGUI: this class contains the main method for running the model in GUI mode
  - NKCOrganization: this class specifies the NKC inner agent, the species of organisation, that works inside a co-evolutionary set
  - Species: this class stores details over all species of organisation
  - XMLReader: this class parses the xml file containing species data

The classes included in this appendix are NKModel, NKFitnessLandscape and NKOrganization, these hope to show the main classes that make up one of the Models constructed.

G.1 File: NKModel.java

```
package NKModel;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import cern.jet.random.Normal;
import cern.jet.random.Uniform;
import cern.jet.random.engine.MersenneTwister;

import CrossModelClasses.ParameterOptions;
import DataCollection.NKDataCollector;
import NKModel.NKFitnessLandscape;
import NKModel.NKOrganization;
import NKModel.OrganizationEdge;

import uchicago.src.sim.engine.BasicAction;
import uchicago.src.sim.engine.Schedule;
import uchicago.src.sim.engine.SimModel;
import uchicago.src.sim.engine.SimModelImpl;
import uchicago.src.sim.network.Edge;
import uchicago.src.sim.network.NetworkFactory;
import uchicago.src.sim.network.Node;


/**
 *
 * Model extends the SimpleModel class of repast in order to set up
 * and run the simulation.  This class continas a method to build
 * the mdoel and three to run the model preStep(), step() and
 * postStep() which occur of every tick of the simulation.
 *
 * @author        Amy Marshall
 * @version       1
 */
public abstract class NKModel extends SimModelImpl implements SimModel
{
        //the local variable of the class
        private NKDataCollector collector;
        private NKFitnessLandscape landscape;
        private int tick_count = 0;
        private Uniform uniform;
        private Normal normal;
        private int[] array_A;
        private double global_maximum_fitness = 0;
        protected String name;
        private Schedule schedule;
        private ArrayList<NKOrganization> agentList;

/***************************************/
/************THE PARAMETERS*************/
/***************************************/

//the numbeer of characteristics an organization has
public int N_size_of;
public void setN_size_of(int N_size_of)
{
        this.N_size_of = N_size_of;
}
public int getN_size_of()
{
        return N_size_of;
}

//the number of characterisitcs each characterisit depends upon
public int K_size_of;
public void setK_size_of(int K_size_of)
{
        this.K_size_of = K_size_of;
}
public int getK_size_of()
{
        return K_size_of;
}

//the number of states each characteristic can have
public int A_size_of;
public void setA_size_of(int A_size_of)
{
        this.A_size_of = A_size_of;
}
public int getA_size_of()
{
        return A_size_of;
}

//the number of orgnaiztions that re thrown onto the landscape
//at the start of the simulation
public int organizations_no_of;
public void setOrganizations_no_of(int organizations_no_of)
{
        this.organizations_no_of = organizations_no_of;
}
public int getOrganizations_no_of()
{
        return organizations_no_of;
}

//the number of decimal places the fitness range goes to
```

```
public int fitness_range_dp;
public void setFitness_range_dp(int fitness_range_dp)
{
        this.fitness_range_dp = fitness_range_dp;
}
public int getFitness_range_dp()
{
        return fitness_range_dp;
}

//whether all K are the same or whether K depends on N
public int K_identical_or_random; // identical (0) or random (1) or
    gaussian (2)
public void setK_identical_or_random(int K_identical_or_random)
{
        this.K_identical_or_random = K_identical_or_random;
}
public int getK_identical_or_random()
{
        return K_identical_or_random;
}

//whether K are neighbours of N or whether K are chosen randomly from N
public int K_neighbours_or_random; // neighbours(0) or random(1)
public void setK_neighbours_or_random(int K_neighbours_or_random)
{
        this.K_neighbours_or_random = K_neighbours_or_random;
}
public int getK_neighbours_or_random()
{
        return K_neighbours_or_random;
}

//whether all A are the same or whether they are dependant on N
public int A_identical_or_random; // identical(0) or random(1)  or
    gaussian (2)
public void setA_identical_or_random(int A_identical_or_random)
{
        this.A_identical_or_random = A_identical_or_random;
}
public int getA_identical_or_random()
{
        return A_identical_or_random;
}

//whether the orgnaiztion can jump over the andscape or not
public int jump_J; // walk(0), local jump(1) or long jump(2)
public void setJump_J(int jump_J)
{
        this.jump_J = jump_J;
}
public int getJump_J()
{
        return jump_J;
}

//the threshold underwhich the orgnaiztion will not move to a fitter
    neighbour
public double fitness_threshold;
public void setFitness_threshold(double fitness_threshold)
{
        this.fitness_threshold = fitness_threshold;
}
public double getFitness_threshold()
{
        return fitness_threshold;
}

//the method by which an oprganisation finds the next neighbour it will
    look at
public int next_neighbour_method; // ordered(0) or random with mem(1) or
    random no mem(2)
public void setNext_neighbour_method(int next_neighbour_method)
{
        this.next_neighbour_method = next_neighbour_method;
}
public int getNext_neighbour_method()
{
        return next_neighbour_method;
}

//the method by which fitness is calculated
public int fitness_method; // average(0) or weakest(1)
public void setFitness_method(int fitness_method)
{
        this.fitness_method = fitness_method;
}
public int getFitness_method()
{
        return fitness_method;
}

//the method by which average fitness is calculated
public int fitness_method_averaging_weightings; // identical(0) or random
    (1)  or gaussian (2)
public void setFiitness_method_averaging_weightings(int
    fitness_method_averaging_weightings)
{
        this.fitness_method_averaging_weightings =
                fitness_method_averaging_weightings;
}
public int getFitness_method_averaging_weightings()
```

```
{
        return fitness_method_averaging_weightings;
}

//the number of successfull jumps allowed
public int jump_successful_limit;
public void setJump_successful_limit(int successful_jump_limit)
{
        this.jump_successful_limit = successful_jump_limit;
}
public int getJump_successful_limit()
{
        return jump_successful_limit;
}

//the umber of unsuccessful jumps allowed
public int jump_search_time_limit;
public void setJump_search_time_limit(int jump_search_time_limit)
{
        this.jump_search_time_limit = jump_search_time_limit;
}
public int getJump_search_time_limit()
{
        return jump_search_time_limit;
}

//the type of walk the orgnaiztion takes over the landscape
public int organization_walk_type; //one_mutant_neighbour(0),
     fitter_dynamics(1), gready_dynamics(2)
public void setOrganization_walk_type(int organization_walk_type)
{
        this.organization_walk_type = organization_walk_type;
}
public int getOrganization_walk_type()
{
        return organization_walk_type;
}

//whether or not the orgnaiztion use a communictaions network to look
//at each others fitnesses and loctaions
public boolean comms_network;
public void setComms_network(boolean comms_network)
{
        this.comms_network = comms_network;
}
public boolean getComms_network()
{
        return comms_network;
}

//whether or not the communictaions network changes over time
public boolean comms_network_change;
public void setComms_network_change(boolean comms_network_change)
{
        this.comms_network_change = comms_network_change;
}
public boolean getComms_network_change()
{
        return comms_network_change;
}

//the chance of the netowke changing over time
public int comms_network_change_chance;
public void setComms_network_change_chance(int comms_network_change_chance
    )
{
        this.comms_network_change_chance = comms_network_change_chance;
}
public int getComms_network_change_chance()
{
        return comms_network_change_chance;
}

//the frequency at which the network changes, every ? ticks
public int comms_network_change_frequency;
public void setComms_network_change_frequency(int
    comms_network_change_frequency)
{
        this.comms_network_change_frequency =
                comms_network_change_frequency;
}
public int getComms_network_change_frequency()
{
        return comms_network_change_frequency;
}

//the type of network used
public int comms_network_type;
public void setComms_network_type(int comms_network_type)
{
        this.comms_network_type = comms_network_type;
}
public int getComms_network_type()
{
        return comms_network_type;
}

//if the netowrk used is a small worl network what is the connect radius
public int comms_network_small_world_connect_radius;
public void setComms_network_small_world_connect_radius(int
    comms_network_small_world_connect_radius)
{
```

```
        this.comms_network_small_world_connect_radius =
                comms_network_small_world_connect_radius;
}
public int getComms_network_small_world_connect_radius()
{
        return comms_network_small_world_connect_radius;
}

//if the netowkr used is a random network what is the connection
    probability
//as a percentage
public int comms_network_connection_probability_percentage;
public void setComms_network_connection_probability_percentage(int
    comms_network_connection_probability_percentage)
{
        this.comms_network_connection_probability_percentage =
                comms_network_connection_probability_percentage;
}
public int getComms_network_connection_probability_percentage()
{
        return comms_network_connection_probability_percentage;
}

//if the network is a small world network was is the rewire probability
//as a percentage
public int comms_network_rewire_probability_percentage;
public void setComms_network_rewire_probability_percentage(int
    comms_network_rewire_probability_percentage)
{
        this.comms_network_rewire_probability_percentage =
                comms_network_rewire_probability_percentage;
}
public int getComms_network_rewire_probability_percentage()
{
        return comms_network_rewire_probability_percentage;
}

//is life and death of orgnaiztions modelled
public boolean life_and_death;
public void setLife_and_death(boolean life_and_death)
{
        this.life_and_death = life_and_death;
}
public boolean getLife_and_death()
{
        return life_and_death;
}

//under what threshold (as a difference between the fitest orgniaztion
//and the orgnaiztion in quetsion) should that orgaiztions die
public double life_and_death_threashold;
public void setLife_and_death_threashold(double life_and_death_threashold)
{
        this.life_and_death_threashold = life_and_death_threashold;
}
public double getLife_and_death_threashold()
{
        return life_and_death_threashold;
}

//what method is used for creating new orngaiztions
public int life_and_death_new_org_method; // random_new_org (0),
    copy_old_org (1), both (2)
public void setLife_and_death_new_org_method(int
    life_and_death_new_org_method)
{
        this.life_and_death_new_org_method = life_and_death_new_org_method
                ;
}
public int getLife_and_death_new_org_method()
{
        return life_and_death_new_org_method;
}

//how is data collected
public int collect_data; //to screen(0), to file(1) or both(2)
public void setCollect_data(int collect_data)
{
        this.collect_data = collect_data;
}

public int getCollect_data()
{
        return collect_data;
}

//what is the name of the file data is collected to
public String data_collection_file_name;
public void setdata_collection_file_name(String data_collection_file_name)
{
        this.data_collection_file_name = data_collection_file_name;
}

public String getdata_collection_file_name()
{
        return data_collection_file_name;
}

//at what number of ticks should the simulation halt?
public int simulation_halt;
public void setsimulation_halt(int simulation_halt)
{
```

```
        this.simulation_halt = simulation_halt;
}

public int getsimulation_halt()
{
        return simulation_halt;
}



public void begin()
{
        buildModel();
    buildSchedule();
}

/**
 * Tears down simulation in preparation for next run, sets
 * back varaibles to reasonable default
 *
 */
public void setup()
{
        //setting everything back to initial values
        collector = null;
        landscape = null;
        tick_count = 0;
        uniform = null;
        normal = null;
        array_A = null;
        global_maximum_fitness = 0.0;
        name = null;
        agentList = null;

        //resetting parameters to their defaults
        fitness_range_dp = 2;
        organizations_no_of = 100;
        N_size_of = 10;
        K_size_of = 8;
        A_size_of = 2;
        comms_network_connection_probability_percentage = 50;
        life_and_death_threashold = 0.1;
        collect_data = 2;
        data_collection_file_name = "./data.txt";
        simulation_halt = 250;
        next_neighbour_method = 0;
        fitness_threshold = 0;

        //setting up a new schedule
        schedule = new Schedule(1);
        agentList = new ArrayList<NKOrganization>();

        //creating the random generators
        MersenneTwister generator1;
        MersenneTwister generator2;
        if(ParameterOptions.SEED == ParameterOptions.DATE)
        {
                Date date1 = new Date();
                generator1 = new MersenneTwister(date1);
                Date date2 = new Date();
            generator2 = new MersenneTwister(date2);
        }
        else
        {
                generator1 = new MersenneTwister(123);
            generator2 = new MersenneTwister(123);
        }

    normal = new Normal(1.0, 1.0, generator1);
    uniform = new Uniform(generator2);

}

/**
 * Creates objects that the simulation uses, agents are created here
 * and added to master list of agents
 *
 */
public void buildModel()
{
        //create the states array
        createA_array();

        //initialize tick count
        tick_count = 0;

        //create the landscape
    landscape = new NKFitnessLandscape(N_size_of, array_A, K_size_of,
            fitness_range_dp, K_identical_or_random,
                        K_neighbours_or_random,
                                fitness_method_averaging_weightings,
                                fitness_method);

        //create all organizations
        createOrganizations(createNetwork());

        //set up the data collector
        collector = new NKDataCollector(this, N_size_of, A_size_of,
                collect_data);
        collector.setUpGraphs();
        collector.setUpDataCollectionToFile(agentList, landscape,
                data_collection_file_name);
```

```
        collector.setUpDataCollection(agentList, landscape);
}


/**
 * Create the array of possible states that each characterisitc in
 * the orgnaiztaion can have
 *
 */
public void createA_array()
{
        //assign states to each characteristic
        array_A = new int[N_size_of];

        if(A_identical_or_random == ParameterOptions.IDENTICAL)
        {//if the number of states of each characteristic is going to be
                identical
                  for(int n = 0; n < N_size_of; n++)
                  {//in each element in the states array put the same number
                          of states
                            array_A[n] = A_size_of;
                  }
        }
        else if(A_identical_or_random == ParameterOptions.RANDOM)
        {//if the number of states is going to be random with a mean of
                A_size_of
                  for(int n = 0; n < N_size_of; n++)
                  {///for each element in the states array create a new
                        random number in this range
                            array_A[n] = uniform.nextIntFromTo(1, A_size_of*2)
                                  ;;
                  }
        }
        else if(A_identical_or_random == ParameterOptions.GAUSSIAN)
        {//if the number of states is going to be random with a gaussian
                shape around A_size_of
                  for(int n = 0; n < N_size_of; n++)
                  {
                          //for each element in the states array create a
                                new gaussian
                          //number and map it to the correct range
                          double w  = normal.nextDouble();
                          //gaussian is from range -3.5 to 3.5, divide by 7
                                to put in
                          //range -0.5 to 0.5
                          double x = w / 7;
                          //add 0.5 to put in range 0 to 1
                          double y = x + 0.5;
                          //multiple by A_size_of*2 to put in the range 0 to
                                2A
                          double z = y*A_size_of*2;
                          //round it to an integer value
                          array_A[n] = (int)Math.round(z);
                  }
        }
}


private void buildSchedule() {

        class NKOrgRun extends BasicAction {
                public void execute() {
                        preStep();
                        step();
                        postStep();
                }
        }

        NKOrgRun run = new NKOrgRun();
    schedule.scheduleActionBeginning(1, run);

    schedule.scheduleActionAt(simulation_halt, this, "stop", Schedule.LAST
          );

        //check that K is smaller than N
        if(K_size_of >= N_size_of)
        {
                //System.err.println("K must be smaller than N");
                //JOptionPane.showMessageDialog(null, "K must be smaller
                      than N", "Loser", JOptionPane.INFORMATION_MESSAGE);
                schedule.scheduleActionAt(0, this, "stop", Schedule.LAST);
        }
        //check the radius of the small worl netowrk is smaller than the
              number of organizations
        if(comms_network_small_world_connect_radius >= organizations_no_of
              )
        {
                //System.err.println("Please ensure the
                      comms_network_small_world_connect_radius " +
                                                        //"is less than
                                                              the number
                                                              of
                                                              organizations
                                                               specified")
                                                              ;
                schedule.scheduleActionAt(0, this, "stop", Schedule.LAST);
        }
}

/**
```

```
 * This method is run before every step takes place, this method will only
       do
 * anyhting in the case that the user has selected to use life and death
       within
 * the simnulation
 *
 */
public void preStep()
{
        if(life_and_death)
        {//if the user has selected to use life and death with in the
              simulation
                while(agentList.size() < organizations_no_of)
                {//while there are less ExtendedNKOrganizations than there
                      are meant to be create a new organziation
                        NKOrganization org = new NKOrganization();
                        createNewOrganizations(org);
                        //add the org to the network if there is one
                        if(comms_network)
                        {
                                addToNetwork(org);
                        }
                        //add the newly created organzition to the agent
                              list
                        agentList.add(org);
                }
        }

        if(comms_network)
        {//if we are communicating with other agents
                for(int i = 0; i < agentList.size(); i++)
                {//each agent must send a communictaion out telling it's
                      fittness
                        NKOrganization org = (NKOrganization) agentList.
                              get(i);
                        org.sendCommunications();
                }
        }
}

/**
 * Iterate through all agents in the simulation and call the method that
 * executes there behaviour
 *
 */
public void step()
{
        int done_walking = 0;
        int done_jumping = 0;
        int size = agentList.size();
    for (int i = 0; i < size; i++)
    {//for every agent in the sinulation
        NKOrganization o = (NKOrganization) agentList.get(i);
        //take an adaptive walk step
        o.adaptiveWalk(null);
        if(!o.getStillWalking())
        {
                done_walking++;
        }
        if(o.reachedJumpLimit())
        {
                done_jumping++;
        }
    }
    if((done_walking >= 100)&&((jump_J == 0)||(done_jumping >= 100)))
    {
        schedule.scheduleActionAt(schedule.getCurrentTime()+1, this, "stop
              ", Schedule.LAST);
    }
    tick_count++;

    //update the graphs
collector.updateGraphs();
}

/**
 * This method is run after every step has taken place, this method will
       only do
 * anyhting in the case that the user has selected to use life and death
       within
 * the simnulation
 *
 * At the end of each tick some ExtendedNKOrganizations will die, this
       method calculates
 * which ExtendedNKOrganizations will be effected by this
 *
 */
public void postStep()
{
        if(life_and_death)
        {//if the user has selected to use life and death
                //System.out.println("life and death");
                //find the maximum fitness
                global_maximum_fitness = 0;
                for(int i = 0; i < agentList.size(); i++)
        {//for everyorganziation in the agent list
                NKOrganization org = (NKOrganization) agentList.get(i);
                if(landscape.getFitness(org.getLocation(), null) >
                      global_maximum_fitness)
                {//if it is the fittest organziation seen so far record
                      its fitness
```

```
                global_maximum_fitness = landscape.getFitness(org.
                    getLocation(), null);
            }
        }

            for(int i = 0; i < agentList.size(); i++)
            {//for every organziation in the agent list
                NKOrganization org1 = (NKOrganization) agentList.
                    get(i);
                if (landscape.getFitness(org1.getLocation(), null)
                    <= global_maximum_fitness -
                    life_and_death_threashold)
                {//see if it's fitness is within the user
                    specified range of the maximum fitness

                    if(comms_network)
                    {//if there are communictaions networks as
                        well as life and death
                        removeLinksInNetwork(org1);
                    }
                    //if its not then remove the rognaization
                        from the agent list
                    agentList.remove(i);
                }
            }
        }

        if(((comms_network)&&(comms_network_change))&&(tick_count ==
            comms_network_change_frequency))
        {//if we are caing the network and we are at the change frequence
            //reset the counter
            tick_count = 0;
            for(int o = 0; o < agentList.size(); o++)
            { //for every orgnaiztions
                //randomly decide if we change this rongaiztions
                    links at al
                int random = uniform.nextIntFromTo(0, 100);
                if(random < comms_network_change_chance)
                {//if changing the orngaiztions links then
                    //run the change netowrk method
                    changeNetwork(o);
                }
            }
        }
}

/**
 * Create all orgnaizations, take in a network of organizations,
 * and pass them all necessary parameters including an initial
 * location
 *
 */
public void createOrganizations(List network)
{
        for(int o = 0; o < network.size(); o++)
        {
            //create an initial location for the organization
            String s = "";
            for (int n = 0; n < N_size_of; n++)
            {
                s = s + uniform.nextIntFromTo(0, array_A[n]-1);
            }
            //create the new organization giving it an initial
                location
            NKOrganization org = (NKOrganization) network.get(o);
            org.setUpOrganization(landscape, s, fitness_threshold,
                jump_J,
                    jump_successful_limit,
                        jump_search_time_limit,
                        comms_network,
                    organization_walk_type,
                            next_neighbour_method);
            //add the organziation to the location list
            agentList.add(org);
        }
}

/**
 * Creates a network of organizations and organization edges, this can
     then
 * be used for organizations to communicate
 *
 * @return List all ndoe sin the network
 */
public List createNetwork()
{
        List network = null;
        //EdgeFactory.createEdge(nodeA, nodeB);
        int size = organizations_no_of;
        Class nodeClass = NKOrganization.class;
        Class edgeClass = OrganizationEdge.class;
        if(comms_network)
        {//if we are using communictaions networks
            if(comms_network_type == ParameterOptions.LINEAR_NETWORK)
            {//if a linear network is to be created
                //create an unlicnked network
                network = NetworkFactory.createUnlinkedNetwork(
                    size, nodeClass);
                for(int i = 0; i < network.size(); i++)
                {//for every node in the network list
                    if(i+1 < network.size())
                    {//if it is not the last node
```

```
                        //connect it both ways with the
                            next node in the list
                        networkLink((NKOrganization)
                            network.get(i), (
                            NKOrganization)network.get(i
                            +1));
                    }
                    else
                    {//if it is the last ndoe in the list
                        //connect it both ways with the
                            first ndoe in the list
                        networkLink((NKOrganization)
                            network.get(i), (
                            NKOrganization)network.get
                            (0));
                    }
            }

    }
    else if(comms_network_type == ParameterOptions.
        FULLY_CONNECTED_NETWORK)
    {//if a fully connected network is to be created
            //create an unlinked network
            network = NetworkFactory.createUnlinkedNetwork(
                size, nodeClass);
            for(int i = 0; i < network.size(); i++)
            {//for every node in the network list
                    for(int j = i; j < network.size(); j++)
                    {//connect it both ways with every other
                        node in the list
                        networkLink((NKOrganization)
                            network.get(i), (
                            NKOrganization)network.get(j
                            ));
                    }
            }
    }
    else if(comms_network_type == ParameterOptions.
        RANDOM_NETWORK)
    {//if a random network is to be created
            double density =
                comms_network_connection_probability_percentage
                /100.0;
            boolean allowLoops = false;
            boolean isSymmetric = true;
            //use the factory method to create this
            network = NetworkFactory.
                createRandomDensityNetwork(size,density,
                allowLoops,isSymmetric,nodeClass,edgeClass);
    }
            else if(comms_network_type == ParameterOptions.
                SMALL_WORLD_NETWORK)
            {//if a small world networkl is to be created
                    int connectRadius =
                        comms_network_small_world_connect_radius;
                    double rewireProb =
                        comms_network_rewire_probability_percentage
                        /100;
                    //use the factory method to create this
                    network = NetworkFactory.
                        createWattsStrogatzNetwork(size,
                        connectRadius, rewireProb, nodeClass,
                        edgeClass);
            }
        }
        else //if not useing communications
        {
                network = NetworkFactory.createUnlinkedNetwork(size,
                    nodeClass);
        }
        return network;
    }

    /**
     * When life and Death is effective this method creates new organizations
     * whilst the model is running dependant on the life_and_death options
     * selected during start up
     *
     * @param organisation that has just been created
     */
    public void createNewOrganizations(NKOrganization org)
    {
        if(life_and_death_new_org_method == ParameterOptions.
            RANDOM_NEW_ORG)
        {//if the new organziations are set to be created randomly
            String s = "";
            for (int n = 0; n < N_size_of; n++)
            {
                s = s + uniform.nextIntFromTo(0, array_A[n]);
            }
            org.setUpOrganization(landscape, s,fitness_threshold,
                jump_J,
                        jump_successful_limit,
                            jump_search_time_limit,
                            comms_network,
                        organization_walk_type,
                            next_neighbour_method);
        }
        else if (life_and_death_new_org_method == ParameterOptions.
            COPY_OLD_ORG)
```

```
{//if the new organziations are set to be created by copying the
    location of an old organziation
        int new_org = uniform.nextIntFromTo(0, agentList.size()-1)
            ;
        NKOrganization old_org = (NKOrganization) agentList.get(
            new_org);
        org.setUpOrganization(landscape, old_org.getLocation(),
            fitness_threshold,
                    jump_J, jump_successful_limit,
                        jump_search_time_limit,
                        comms_network,
                    organization_walk_type,
                        next_neighbour_method);
}
else if (life_and_death_new_org_method == ParameterOptions.BOTH)
{//if both (Levinthal)
        double average_fitness = 0;
for(int i = 0; i < agentList.size(); i++)
{//for every ExtendedNKOrganization in the agent list
        NKOrganization o = (NKOrganization) agentList.get(i);
                average_fitness = average_fitness + landscape.
                    getFitness(o.getLocation(), null);
}
average_fitness = average_fitness / agentList.size();
        double genetic_load = 1 - average_fitness/
            global_maximum_fitness;
        double new_org_type = uniform.nextDouble();
        if(new_org_type <= genetic_load)
        {
                int new_org = uniform.nextIntFromTo(0, agentList.
                    size()-1);
                NKOrganization old_org = (NKOrganization)
                    agentList.get(new_org);
                org.setUpOrganization(landscape, old_org.
                    getLocation(),fitness_threshold,
                                jump_J, jump_successful_limit,
                                    jump_search_time_limit,
                                    comms_network,
                                organization_walk_type,
                                    next_neighbour_method);
        }
        else
        {
                String s = "";
                for (int n = 0; n < N_size_of; n++)
                {
                        s = s + uniform.nextIntFromTo(0, array_A[n
                            ]);
                }
                org.setUpOrganization(landscape, s,
                    fitness_threshold, jump_J,
                                jump_successful_limit,
                                    jump_search_time_limit,
                                    comms_network,
                                organization_walk_type,
                                    next_neighbour_method);
        }
}
}

/**
 * Add new links within the network for the new orgnaization
 *
 * @param org
 */
private void addToNetwork(NKOrganization org)
{
        if((comms_network_type == ParameterOptions.LINEAR_NETWORK)
                ||(comms_network_type == ParameterOptions.
                    SMALL_WORLD_NETWORK))
        {
                //add two random two way network links
                int random_link_one = uniform.nextIntFromTo(0, agentList.
                    size()-1);
                int random_link_two = uniform.nextIntFromTo(0, agentList.
                    size()-1);

                //adding the first line
                networkLink(org, (NKOrganization)agentList.get(
                    random_link_one));
                networkLink(org, (NKOrganization)agentList.get(
                    random_link_two));
        }
        else if(comms_network_type == ParameterOptions.
            FULLY_CONNECTED_NETWORK)
        {
                //add a link in the network between this organization and
                    every other organization
                for(int o = 0; o < agentList.size(); o++)
                {//for every orgnaization already in the agent list
                        //add a two way network link
                        networkLink(org, (NKOrganization)agentList.get(o))
                            ;
                }
        }
        else if(comms_network_type == ParameterOptions.RANDOM_NETWORK)
        {
                for(int o = 0; o < agentList.size(); o++)
                {//for every orgnaization already in the agent list
                        //pick a random number between 0 and 100
                        int yes_or_no = uniform.nextIntFromTo(0, 100);
```

```
                        if(yes_or_no >
                                comms_network_connection_probability_percentage
                                )
                        {//if this is within random then create an edges
                                to the next node
                                  networkLink(org, (NKOrganization)agentList
                                          .get(o));
                        }
                }
        }
}

/**
 * Creates a link in the communictaions network between org1 and org2
 *
 * @param org1
 * @param org2
 */
private void networkLink(NKOrganization org1, NKOrganization org2)
{
        //creates the nodes and the edge between them
        NKOrganization fromNode1;
        NKOrganization toNode1;
        OrganizationEdge edge1;

        //sets up the edge one way
        fromNode1 = org1;
        toNode1 = org2;
        edge1 = new OrganizationEdge();
        edge1.setFrom(fromNode1);
        edge1.setTo(toNode1);
        toNode1.addInEdge(edge1);
        fromNode1.addOutEdge(edge1);

        //creates the nodes and the edge between them
        NKOrganization fromNode2;
        NKOrganization toNode2;
        OrganizationEdge edge2;

        //sets up the sege the other way
        fromNode2 = org2;
        toNode2 = org1;
        edge2 = new OrganizationEdge();
        edge2.setFrom(fromNode2);
        edge2.setTo(toNode2);
        toNode2.addInEdge(edge2);
        fromNode2.addOutEdge(edge2);
}

/**
 * The given orgnaiztaion is dead, remove all the links the given
 * organization has in the network
 *
 * @param org1
 */
private void removeLinksInNetwork(NKOrganization org1)
{
        for(int i = 0; i < agentList.size(); i++)
        {//for all organiztaion in the simulation
                NKOrganization org2 = (NKOrganization) agentList.get(i);
                if(org2.hasEdgeFrom(org1))
                {//if that orgnaiztaion has an edge from the one we are
                        removing
                          ArrayList inEdges = org2.getInEdges();
                          for(int edge_index = 0; edge_index < inEdges.size
                                  (); edge_index++)
                          {//look at all edges
                                  Edge e = (Edge) inEdges.get(edge_index);
                                  if(e.getFrom() == org1)
                                  {//find that edge and remove it
                                          org2.removeInEdge(e);
                                  }
                          }
                }
                if(org2.hasEdgeTo(org1))
                {//if that organization has an edge to the one we are
                        removing
                          ArrayList outEdges = org2.getOutEdges();
                          for(int edge_index = 0; edge_index < outEdges.size
                                  (); edge_index++)
                          {//look at all edges
                                  Edge e = (Edge) outEdges.get(edge_index);
                                  if(e.getTo() == org1)
                                  {//find that edge and remove it
                                          org2.removeOutEdge(e);
                                  }
                          }
                }
        }
}

/**
 * Change the links the given organization has within the network
 *
 * @param organization_index
 */
private void changeNetwork(int organization_index)
{
        //CHANGES MADE WILL DEGRADE SMALLWORLD AND LINEAR NETWORKS SO THEY
                NO LONGER
        //TAKE THERE ORIGIONAL FORM
```

```
        NKOrganization org = (NKOrganization) agentList.get(
                organization_index);
        ArrayList<Edge> outEdges = org.getOutEdges();
        if(((comms_network_type == ParameterOptions.RANDOM_NETWORK)
                        ||(comms_network_type == ParameterOptions.
                                LINEAR_NETWORK))
                        ||(comms_network_type == ParameterOptions.
                                SMALL_WORLD_NETWORK))
        {//if we are looking at a random network or a linear network
                for(int e = 0; e < outEdges.size(); e++)
                {//for every out edge of this orgnaization
                        //decide randomly whether we are going to change
                                it
                        int random_edge_change = uniform.nextIntFromTo(0,
                                100);
                        if(random_edge_change <
                                comms_network_change_chance)
                        {//if we are going to change it
                                //remove that edge
                                Edge old_edge = (Edge) outEdges.get(e);
                                Node other_org = (Node) old_edge.getTo();
                                org.removeOutEdge(old_edge);
                                org.removeInEdge(old_edge);
                                other_org.removeOutEdge(old_edge);
                                other_org.removeInEdge(old_edge);

                                //decide which orgnaization we are
                                        creating a new edge with
                                int random_new_edge = uniform.
                                        nextIntFromTo(0, agentList.size()-1)
                                        ;
                                //add the new edge if the organization is
                                        not trying to link to itself
                                if(random_new_edge != e)
                                {
                                        networkLink(org, (NKOrganization)
                                                agentList.get(
                                                random_new_edge));

                                }
                        }
                }
        }
        else //if(comms_network_type == ParameterOptions.
                FULLY_CONNECTED_NETWORK)
        {
                //nothing changes becasue it will always be a fully
                        connected network
        }
}
```

```
/**
 * Returns the model parameters
 *
 */
public String[] getInitParam()
{
        String[] params = new String[] {"N_size_of",
                                "K_size_of",
                                "fitness_range_dp",
                                "A_identical_or_random",
                                "jump_J",
                                "A_size_of",
                                "K_identical_or_random",
                                "K_neighbours_or_random",
                                "next_neighbour_method",
                                "fitness_method",
                                "organizations_no_of",
                                "fitness_threshold",
                                "jump_successful_limit",
                                "jump_search_time_limit",
                                "organization_walk_type",
                                "fitness_method_averaging_weightings",
                                "comms_network",
                                "comms_network_type",
                                "comms_network_change",
                                "comms_network_change_chance",
                                "comms_network_change_frequency",
                                "comms_network_small_world_connect_radius",
                                "life_and_death",
                                "life_and_death_threashold",
                                "life_and_death_new_org_method",
                                "comms_network_connection_probability_percentage",
                                "comms_network_rewire_probability_percentage",
                                "collect_data",
                                "data_collection_file_name",
                                "simulation_halt"};
        return params;
}

/**
 * Returns the model name
 */
public String getName()
{
        return name;
}

/**
 * Returns the model schedule
 */
public Schedule getSchedule()
```

```
        {
                return schedule;
        }
}
```

G.2 File: NKFitnessLandscape.java

```
package NKModel;

import java.util.Date;

import CrossModelClasses.AbstractFitnessLandscape;
import CrossModelClasses.ParameterOptions;

import cern.jet.random.Normal;
import cern.jet.random.Uniform;
import cern.jet.random.engine.MersenneTwister;

/**
 * This is the NKFitnessLandscape and it stores the relative fitnesses
 * of locations in the landscape such that orgnaizations on this landscape
 * can calculate their fitness.
 *
 * @author Amy Marshall
 *
 */

public class NKFitnessLandscape extends AbstractFitnessLandscape
{
        /**
         * Constructor creates the fitness landscape assigning user parameters to
         * variables of the ladnscape
         *
         * @param N_size_of the number of characteristics in an orgnaiztaion
         * @param array_A the number of states each N hass to choose form
         * @param K_size_of the number of characteristics each N depends on
         * @param fitness_range_dp the decimal places the fitness range goes to
         * @param K_identical_or_random if every N has the same K or K is assigned
                randomly to N
         * @param K_neighbours_or_random wheteher K are neighbours to N or
                assigned randomly
         * @param fitness_method_averaging_weightings whether averaging should be
                IDENTICAL or WEIGHTED
         * @param fitness_method whether fitness should be calculated using the
                AVERAGE or WEAKEST
         */
        public NKFitnessLandscape(int N_size_of, int[] array_A, int K_size_of, int
                fitness_range_dp,
                int K_identical_or_random, int K_neighbours_or_random, int
                        fitness_method_averaging_weightings,
                int fitness_method)
{
        //asigning user parameters
        this.N_size_of = N_size_of;
        this.array_A = array_A;
        this.K_size_of = K_size_of;
        this.fitness_range_dp = fitness_range_dp;
        this.K_identical_or_random = K_identical_or_random;
        this.K_neighbours_or_random = K_neighbours_or_random;
        this.fitness_method_averaging_weightings =
              fitness_method_averaging_weightings;
        this.fitness_method = fitness_method;

        //setting up the random generators
        MersenneTwister generator1;
        MersenneTwister generator2;
        if(ParameterOptions.SEED == ParameterOptions.DATE)
        {
                Date date1 = new Date();
                generator1 = new MersenneTwister(date1);
                Date date2 = new Date();
            generator2 = new MersenneTwister(date2);
        }
        else
        {
                generator1 = new MersenneTwister(123);
            generator2 = new MersenneTwister(123);
        }

    normal = new Normal(1.0, 1.0, generator1);
    uniform = new Uniform(generator2);

    //creating the array for distribution of K
        array_K = createKList();
        //creating the weighting calculations
        weightings = createWeightingsList();
}


/**
 * Use the detail given to work out the fitness of this location
 *
 * @param fitness_method                        characteristic
      dependancies are averaged, weakest taken or weighted
 * @param uniform_or_random_K               number of dependancies of is K the
       average number is K
 * @param neighbours_or_random_K            is K made up of the closest
      characteristics to N or is K random
 */
```

```
public double getFitness(String key, String[] locations)
{
        int[] state_array = stringToArray(key);

        double average_fitness = 0.0;
        double weakest_fitness = 1.0;
        for(int n = 0; n < N_size_of; n++)
        {//for every characteristic in the location
                int[] K = new int[array_K[n].length+1];
                //create a state array of the states of characteristics
                      that effect them
                K[0] = state_array[n];
                for(int k = 0; k < array_K[n].length; k++)
                {//for each characteristic that effects N
                        //add it to N's state array
                        K[k+1] = state_array[array_K[n][k]];
                }
                double new_fitness = characteristicFitness(n,K);

                //save the average fitness
                average_fitness = average_fitness + new_fitness;

                //save the weakest fitness
                if(new_fitness < weakest_fitness)
                {
                        //save this as the weaker fitness
                        weakest_fitness = new_fitness;
                }
        }
        if(fitness_method == ParameterOptions.AVERAGE)
        {
                //divide through by N to calc average fitness
                if(fitness_method_averaging_weightings == ParameterOptions
                      .IDENTICAL)
                {//if we are using no weightings return the average of the
                       fitness
                        return average_fitness / N_size_of;
                }
                else //if(fitness_method_averaging_weightings ==
                      ParameterOptions.RANDOM)
                {
                        //if we are using weightings they have already
                              been weighted
                        //to be a fraction of the overall fitness
                        return average_fitness;
                }
        }
        else //if (fitness_method == ParameterOptions.WEAKEST)
        {
                return weakest_fitness;
        }
        }

}
```

G.3 File: NKOrganization.java

```
package NKModel;

import java.util.ArrayList;
import java.util.Date;

import CrossModelClasses.ParameterOptions;
import CrossModelClasses.AbstractOrganization;
import cern.jet.random.Uniform;
import cern.jet.random.engine.MersenneTwister;

import uchicago.src.sim.network.Edge;
import uchicago.src.sim.network.Node;



/**
 * The organization class is an agent that moves from location to location
 * across the landscape (moving only when the next location found is fitter)
 *
 * @author        Amy Marshall
 * @version       1
 */

public class NKOrganization extends AbstractOrganization implements Node
{

        //local variables passed in from the Model
        private int jump_J;
        private int jump_successful_limit;
        private int jump_search_time_limit;
        private boolean communications;

        //monitors how many successful jumps have been taken
        private int successfulJumps;
        private int unsuccessfulJumps;

        private boolean still_walking;
        //private boolean still_jumping;

        private ArrayList<Edge> inEdges;
        private ArrayList<Edge> outEdges;
        private String nodeLabel;
```

```
public NKOrganization()
{
        //set to true as the orngaiztion has only just been created
        //and thus will still be moving
        still_walking = true;
        //still_jumping = true;

        //creating the random genorators
        MersenneTwister generator2;
        if(ParameterOptions.SEED == ParameterOptions.DATE)
        {
                Date date = new Date();
                generator2 = new MersenneTwister(date);
        }
        else
        {
                generator2 = new MersenneTwister(123);
        }
    uniform = new Uniform(generator2);

        inEdges = new ArrayList<Edge>();
        outEdges = new ArrayList<Edge>();
        nodeLabel = "";
}

/**
 * Set up the orgnaiztion, called when the organiztion is created
 * to give it all the detail needed to walka over the landscape
 *
 * @param landscape the fitness landscape belonging to the orgnaiztion
 * @param location_key the current location of the orgnaiztion
 * @param fitness_threshold under which the orgnaiztion wont move
 * @param jump_J whether or not the orngaiztion will jump
 * @param jump_successful_limit number of successful jumps allowed
 * @param jump_search_time_limit number of failed jump attempts allowed
 * @param communications whether or not the organization can communicate
       with others
 * @param organizational_walk_type the way the orgnaiztion walks over the
       landscape
 */
public void setUpOrganization(NKFitnessLandscape landscape, String
      location_key,
                double fitness_threshold, int jump_J, int
                      jump_successful_limit,
                int jump_search_time_limit, boolean communications,
                int organizational_walk_type, int next_neighbour_method)
{
        //setting initial location on landscape
        this.landscape = landscape;
        this.location_key = location_key;
        location_fitness = landscape.getFitness(location_key, null);
        //setting the parameters passed through from the model
        this.fitness_threshold = fitness_threshold;
        this.jump_J = jump_J;
        this.jump_successful_limit = jump_successful_limit;
        this.jump_search_time_limit = jump_search_time_limit;
        this.communications = communications;
        this.organizational_walk_type = organizational_walk_type;
        this.next_neighbour_method = next_neighbour_method;
        //setting up local variables
        ticksToFindFitterVariant = 0;
        ticksSinceLastMove = 0;

}

/**
 * Records whether or not the orgnaiztaion is still walking (this is used
 * by the data collection class)
 *
 * @return true is the orgnaiztaion is still walkling, flase otherwise
 */
public boolean getStillWalking()
{
        return still_walking;
}


/**
 * Returns whether or not the organisation has finished jumping
 *
 * @return true is the number of jumps taken is equal to the limit
 */
public boolean reachedJumpLimit()
{
        if((successfulJumps == jump_successful_limit)&&(
                jump_successful_limit != 0))
        {
                return true;
        }
        else
        {
                return false;
        }
}

/**
 * Move the organization to this location
 *
 * @param location_key  location to move to
 */
public void moveTo(String location_key, String[] C_locations)
```

```
{
        super.moveTo(location_key, C_locations);
        unsuccessfulJumps = 0;
        still_walking = true;
}

/**
 * Returns the current fitness of the organization
 *
 * @return Double
 */
public Double getFitness(String[] C_locations)
{
        return location_fitness;
}

/**
 * Every step of the simulation the organization will attempt to take
 * a step on its adaptive walk.
 *
 * If communication is activaed the organization will first attemtp to
 * communicate with others in its network to determin if it can move
 * to another place on the landscape through the network
 *
 * Otheriwse this will be a step across the landscape to a one mutant
 * neighbour if there is a fitter one, and a jump if not (IFF jumps are
 * activated)
 *
 */
public void adaptiveWalk(String[] C_locations)
{
        ticksSinceLastMove++;
        //The organization tries to communicate with others in its network
        if(communications)
        {//if communications are activated
                if(searchCommunications())
                {//if there is a fitter option within the network
                        //dont continue
                        return;
                }
        }

        //if there isn't a fitter alternative in the network or
            communication
        //are not activated

        if(nearestNeighbours == null)
        {//find the nearest neighbours of the current location
                nearestNeighbours = landscape.getAllNeighbours(
                        location_key);
        }

        if(step())
        {//if a step across the landscape is taken return and do not
            attempt to jump
                return;
        }

//if we have looked at all the fitter neighbours
        //System.out.println("successfulJumps: " + successfulJumps);
        if((((successfulJumps < jump_successful_limit)||(
            jump_successful_limit == 0))&&(jump_J == 1))
                &&((jump_search_time_limit == 0)||(unsuccessfulJumps <
                    jump_search_time_limit)))
        {//if the organization has made fewer successful jumps than the
            maximum
                //assk the current location for a long jump
                String jump = landscape.getLongJump(jump_search_time_limit
                    );
        if(jump != null)
        {//if a jump is returned
                if(landscape.getFitness(jump, null) - getFitness(null) >
                        fitness_threshold)
                {//if the new location is fitter than the current location
                     then move
                        moveTo(jump, null);
                        still_walking = true;
                        //System.out.println("successfulJumps = " +
                            successfulJumps );
                        successfulJumps++;
                        return;
                }
                else
                {//if the new location is less fit that the current
                    location
                        unsuccessfulJumps++;
                }
        }
        }

}

/**
 * Attempts to take a step across the landscape, return true if a step is
 * made and false otherwise
 *
 * @param C_locations lists current locations of other species of
        orgnaization
 * @return true if a step acros the landscape is taken and flase otherwise
 */
private boolean step()
{
```

```
if(organizational_walk_type == ParameterOptions.
    ONE_MUTANT_NEIGHBOUR)
{//on each tick look at one neighbour see if it is fitter and move
    to it if it is
     if(nextNeighbour < nearestNeighbours.length)
{//if there is another nearest neighbour available
                String neighbour = null;
                neighbour = getNextNeighbour(next_neighbour_method
                    );
                if(neighbour != null)
                {//neighbour will return null if there are no more
                    spaces to check for fitter neighbours
                    if(landscape.getFitness(neighbour, null) -
                        getFitness(null) >
                        fitness_threshold)
                    {//if the next neighbour returned is
                        fitter the that current position
                        moveTo(neighbour, null);
                        return true;
                    }
                }
                else
                {//if there are no more spaces left to check then
                    nextNeighbour = nearestNeighbours.length;
                    still_walking = false;
                }
}
        else
        {//otherwise stop walking
                still_walking = false;
        }
}
else if (organizational_walk_type == ParameterOptions.
    FITTER_DYNAMICS)
{//on each tick look at all neighbours and move to one of the
    fittest ones
        double fittest_location_value = location_fitness;
        //create an array list to store all locations found with
            the max
        //fittness (if there is more than one location
        ArrayList<String> same_fitness = new ArrayList<String>();
        //add the fitness of the current location to this
        same_fitness.add(location_key);

        for (int i = 0; i < nearestNeighbours.length; i++)
        {//for every neighbour to this location
                //look at the next neighbour
                String next_location = nearestNeighbours[i];
                if(next_location != null)
                {//if the next neighbour is not equal to null
                        //find its fitness
                        double next_location_value = landscape.
                            getFitness(next_location, null);
                        if(next_location_value >
                            fittest_location_value)
                        {//if the fitness is higher than the
                            current fitteset
                            //clear the array list
                            same_fitness.clear();
                            //add the new loction
                            same_fitness.add(next_location);
                            //update the fittest value
                            fittest_location_value =
                                next_location_value;
                        }
                        else if (next_location_value ==
                            fittest_location_value)
                        {//if the location has the same value as
                            the current fittest
                            same_fitness.add(next_location);
                        }
                }
        }

        if(!(same_fitness.get(0)).equals(location_key))
        {//if the highest fitness isn't this location
                //randomly choose one of the locations with
                    highest fitness
                int step = uniform.nextIntFromTo(0, same_fitness.
                    size() - 1);
                //move to that location
                moveTo(same_fitness.get(step), null);
                return true;
        }
        else
        {//otherwise stop walkling
                still_walking = false;
        }
}
else if (organizational_walk_type == ParameterOptions.
    GREEDY_DYNAMICS)
{//on each tick look at neighbours in turn until one is found that
    is fitter and move to that one
        for (int i = 0; i < nearestNeighbours.length; i++)
        {//for every neighbour of this location
                String next_neighbour = nearestNeighbours[i];
                if(next_neighbour != null)
                {// the neighbour is not empty
                        //find its fitness
                        Double next_neighbour_fitness = landscape.
                            getFitness(next_neighbour, null);
```

```
                        if(next_neighbour_fitness >
                            location_fitness)
                        {//if its fitness is higher than the
                            cururent fitness move
                            moveTo(next_neighbour, null);
                            return true;
                        }
                    }
                }
                //if no higher fitness is found stop walkling
                still_walking = false;
            }
            //if no the method arrives here no move has been made so return
                false
            return false;
    }


    /**
     * Sending a communication out through the edges of the netwokr
     *
     */
    public void sendCommunications()
    {
            for (int i = 0; i < outEdges.size(); i++)
            {//for every edge going out to an organziation
                    OrganizationEdge e = (OrganizationEdge) outEdges.get(i);
                    //leave a message for that organzitaion to pick up
                    e.leaveFitnessMessage(location_fitness,location_key);
            }
    }

    /**
     * Looking at all other agents communications through the edges of the
        network
     *
     * @return true if the organization moved to another location using one of
         these
     * communications false otherwise
     */
    private boolean searchCommunications()
    {
            String best_location = location_key;
            double best_fitness = location_fitness;
            for (int i = 0; i < inEdges.size(); i++)
            {//for every edge going into an organization
                    OrganizationEdge e = (OrganizationEdge) inEdges.get(i);
                    //collect the message that organziation left
                    FitnessMessage msg = e.readFitnessMessage();
                    double edge_fitness = msg.getFitness();
                    String edge_location = msg.getLocation();
                    //if it has a better fitness than the current orgnaiztaion
                        or any
                    //other linked orgnaization whoes message has been read so
                        far
                    if(edge_fitness > best_fitness)
                    {//then save the details of the loction this orgnaiztaion
                        is at
                            best_location = edge_location;
                            best_fitness = edge_fitness;
                    }
            }
            if(!best_location.equals(location_key))
            {//if there is a location that is better than the current one move
                to it
                    moveTo(best_location, null);
                    //return true so that the will not be another move this
                        time step
                    return true;
            }
            //return flase so that an adaptive walk will be attempted this
                time step
            return false;
    }

    /**
     * Part of Node interface, add an edge going into this node
     * this edge is used to collect information
     *
     */
    public void addInEdge(Edge edge) {
            inEdges.add(edge);
    }

    /**
     * Part of Node interface, add an out edge coming form this node,
     * this edge is used to send information
     */
    public void addOutEdge(Edge edge) {
            outEdges.add(edge);
    }

    /**
     * Part of Node interface, remove all in edges
     */
    public void clearInEdges() {
            inEdges.clear();
    }

    /**
     * Part of Node interface, remove all out edges
```

```
 */
public void clearOutEdges() {
        outEdges.clear();
}

/**
 * NOT USED, IMPLIMENT LATER IF NEEDED
 */
public Object getId() {
        return null;
}

/**
 * Part of Node interface, return an array list of all in edges
 */
public ArrayList<Edge> getInEdges() {
        return inEdges;
}

/**
 * Part of Node interface, return the node label
 */
public String getNodeLabel() {
        return nodeLabel;
}

/**
 * Part of Node interface, return an array list of all out edges
 */
public ArrayList<Edge> getOutEdges() {
        return outEdges;
}

/**
 * Part of Node interface, return true if there is an in edge from this
 * node to the inputted node
 *
 * @param Node node
 */
public boolean hasEdgeFrom(Node node) {
        for(int i = 0; i < inEdges.size();i++)
        {
                Edge e = (Edge) inEdges.get(i);
                Node n = e.getFrom();
                if(n == node)
                {
                        return true;
                }
        }
        return false;
}

/**
 * Part of Node interface, return true if there is an out edge from this
 * node to the inputted node
 *
 * @param Node node
 */
public boolean hasEdgeTo(Node node) {
        for(int i = 0; i < outEdges.size();i++)
        {
                Edge e = (Edge) outEdges.get(i);
                Node n = e.getTo();
                if(n == node)
                {
                        return true;
                }
        }
        return false;
}

/**
 * Part of Node interface, remove the specified edge from the in edge
 * array list
 *
 * @param Edge edge
 */
public void removeInEdge(Edge edge) {
        inEdges.remove(edge);
}

/**
 * Part of Node interface, remove the specified edge from the out edge
 * array list
 *
 * @param Edge edge
 */
public void removeOutEdge(Edge edge) {
        outEdges.remove(edge);
}

/**
 * Part of Node interface, set the node label to the give string
 *
 * @param String label
 */
public void setNodeLabel(String label) {
        nodeLabel = label;
}

}
```

Bibliography

AgentSheets (2006), 'Agentsheets', [online]. Available from: http://www.agentsheets.com/, [Last accessed: 18/03/2007].

Altaweel, M., Collier, N., Howe, T., Najlis, R., North, M., Parker, M., Tatara, E. & Vos, J. R. (n.d.), 'Repast: Recursive porus agent simulation toolkit', [online]. Available from: http://repast.sourceforge.net/, [Last accessed: 18/03/2007].

Anderson, P. (1999), 'Complexity theory and organizational science', *Organizational Science* **10**(3), 216–232.

Argote, L. (2000), 'Knowledge transfer: A basis for competitive advantage in firms', *Organizational Beahvior and Human Decision Processes* **82**(1), 150–169.

Balan, G. C., Cioffi-Revilla, C., Luke, S., Panait, L. & Paus, S. (2003), Mason: A java multi-agent simulation library, *in* 'Agent 2003 Conference on Challenges in Social Simulation', Argonne National Laboratory, the University of Chicago, Chicago, Illinois, pp. 49–64.

Balan, G. C., Luke, S. & Panait, L. (n.d.), 'Mason', [online]. Available from: http://cs.gmu.edu/ eclab/projects/mason/, [Last accessed: 18/03/2007].

Bellifemine, F., Caire, C., Rimassa, G., Poggi, A. & Trucco, T. (2000), 'Jave agent development framework', [online]. Available from: http://jade.tilab.com/, [Last accessed: 18/03/2007].

Bianchi, G., Piccolo, F. L. & Salsano, S. (2006), 'Measurement study of the mobile agent jade platform', *wowmom* **0**, 638–646.

Brassel, K. H. (n.d.), 'Versitile simulation environment for the internet', [online]. Available from: http://www.vseit.de/VSEit09/index.html, [Last accessed: 18/03/2007].

Brooks, J. (1996), 'Cscw as form of organizational momory: Implications for organizational learning', *SIGOIS Bulletin* **17**(3), 39–42.

Bryson, J. (2002), The behavior-oriented design of modular agent intelligence., *in* 'Agent Technologies, Infrastructures, Tools, and Applications for E-Services', Vol. 2592 of *Lecture Notes in Computer Science*, Springer, pp. 61–76.

Bryson, J. (2005), 'Bod mason', [online]. Available from: http://www.cs.bath.ac.uk/%7Ejjb/web/BOD/BOD-MASON.html, [Last accessed: 18/03/2007].

Burton, R. M. & Carroll, T. (2000), 'Organizations and complexity: Searching for the edge of chaos', *Computational and Mathemtica Organizational Theory* **6**(4), 319–337.

Carroll, L. (1865), *Through the Looking Glass*, London: MacMillan.

Clancey, W. J. (1995), A tutorial on situated learning, *in* J. Sel, ed., 'Proceedings of the International Conference on Computers and Education', AACE, Charlottesville, VA, pp. 49–70.

CollabNet (2006), 'Tigris.org: Open software engineering tools', [on line]. Available from: http://quicksilver.tigris.org/, [Last accessed: 18/03/2007].

Collier, N. (2000), Repast: An extensible framework for agent simulation, Technical report, Social Science Research Computing, University of Chicago, Chicago, Illinois.

Detlor, B. & Serenko, A. (2002), 'Agent toolkits: A general overview of the market and an assessmentof instructor satisfaction with utilizing toolkits in the classroom', *ACM Trans. Model. Comput. Simul.* **16**(1), 1–25.

Gao, Y., Madey, G. & Xu, J. (2003), A docking experiment: Swarm and repast for social modeling, *in* 'paper presented at the Seventh Annual Swarm Researchers, Notre Dame, IN.'.

Group, G. T. (2007), 'Monte carlo simulation software for decision and risk analysis', [on line]. Available from: http://www.goldsim.com/, [Last accessed: 18/03/2007].

Group, T. A. O. S. (2006), 'What is jack?', [online]. Available from: http://www.agent-software.com.au/jack.html, [Last accessed: 18/03/2007].

Gutknecht, O., Ferber, J. & Michel, F. (2002), 'The madkit project', [online]. Available from: http://www.madkit.org/, [Last accessed: 18/03/2007].

Hofmann, C. & Tobias, R. (2004), 'Evaluation of free java-libraries for social-scientific agent based simulation.', *J. Artificial Societies and Social Simulation*.

Huber, M. J. (2001), 'Intelligent reasoning systems', [online]. Available from: http://www.marcush.net/IRS/irs_downloads.html, [Last accessed: 18/03/2007].

Institute, T. B. (2000), 'Ascape', [online]. Available from: http://www.brookings.edu/es/dynamics/models/ascape/main.htm, [Last accessed: 18/03/2007].

Kauffman, S. A. (1993), *The Origins of Order: Self Organization and Selection in Evolution*, Oxford University Press, New York, NY.

Kauffman, S. A. (1995), 'Technology and evolution: escaping the red queen effect', *The McKinsey Quarterly* (1), 118–129.

Klein, J. (n.d.), 'Breve: a 3d simulation environment for multi-agent simulations and artificial life', [online]. Available from: http://www.spiderland.org/breve, [Last accessed: 18/03/2007].

Lazer, D. & Freidman, A. (2005), The parable of the hare and the tortoise: Small worlds, diversity, and system performance, KSG Working Paper RWP05-058, John F. Kennedy School of Government, Harvard University. Available at SSRN: http://ssrn.com/abstract8̄32627.

Levinthal, D. A. (1997), 'Adaptation on rugged landscapes', *Management Science* **43**(7), 934–950.

Levitan, B., Lobo, J., Schuler, R. & Kauffman, S. (2002), 'Evolution of organizational performance and stability in a stochastic environment', *Computational and Mathematical Organization Theory* **8**(4), 281–313.

Macal, C. & North, M. (2005), Tutorial on agent-based modeling and simulation, *in* 'WSC '05: Proceedings of the 37th conference on Winter simulation', Argonne National Laboratory, Winter Simulation Conference, Orlando, Florida, pp. 2–15.

McCarthy, I. P. (2002), Manufacturing fitness and nk models, *in* 'Manufacturing Complexity Network Conference', Downing College, University of Cambridge,, Cambridge, UK, pp. 27–40.

McKelvey, B. (1999), 'Avoiding complexity catastrophe in coevolutionary pokets: Strategies for rugged landscapes', *Organization Science* **10**(3), 294–321.

McKelvey, B. & Yuan, Y. (2004), 'Situated learning theory: Adding rate and complexity effects via kauffman's nk model', *Nonlinear Dynamics and Life Sciences* **8**(1), 65–102.

NetLogo (2007), 'Netlogo educators', [online]. Available from: http://groups.yahoo.com/group/netlogo-educators/, [Last accessed: 18/03/2007].

NetLogo (n.d.), 'Netlogo users', [online]. Available from: http://groups.yahoo.com/group/netlogo-users, [Last accessed: 18/03/2007]/.

Parker, M. T. (2001), 'What is ascape and why should you care?', *J. Artificial Societies and Social Simulation*.

Poli, R. & Sloman, . (1996), Sim_agent: A toolkit for exploring agent designs., *in* M. Wooldridge, J. P. Müller & M. Tambe, eds, 'ATAL', Vol. 1037 of *Lecture Notes in Computer Science*, Springer, pp. 392–407.

Repenning, A. (2000), Agentsheets: an interactive simulation environment with end-user programmable agents, *in* 'Interaction 2000', Tokyo, Japan.

Rivkin, J. W. (2000), 'Imitation of complex strategies', *Management Science* **46**(6), 824–844.

Rivkin, J. W. & Siggelkow, N. (2003), 'Balancing search and stability: Interdependencies among elements of organizational design', *Management Science* **49**(3), 290–311.

Russell, S. & Norvig, P. (1995), *Artificial Intelligence: A Modern Approach*, Prentice Hall.

Solow, D. (2003), 'Mathematical models for studying the value of cooperational leadership in team replacement', *Computational and Mathematical Organizational Theory* **9**(1), 61–81.

Solow, D., Burnetas, A., Roeder, T. & Greenspan, N. S. (1999), 'Evolutionary consequences of selected locus-specific variations in epistasis and fitness constribution in kauffman's nk model', *Journal of Theoretical Biology* **196**, 181–196.

Solow, D., Piderit, S., Burnetas, A. & Leenawong, C. (2005), 'Mathematical models for studying the value of motivational leadership in teams', *Computational and Mathematical Organization Theory* **11**(1), 5–36.

Solow, D., Vairaktarakis, G., Piderit, S. K. & Tsai, M. (2002), 'Managerial insights into the effects of interactions on replacing members of a team', *Management Science* **48**(8), 1060–1073.

Sonnessa, M. (2004), 'Jas: Java agent-based simulation library. an open framework for algorithm-intensive simulations.'.

Sonnessa, M., Boero, R., Ferraris, G. & Richiardi, M. (2004), 'Java agent based simulation library', [online]. Available from: http://jaslibrary.sourceforge.net/projects.html, [Last accessed: 18/03/2007].

StarLogo (2000), 'Starlogo on the web', [on line]. Available from: http://education.mit.edu/starlogo/, [Last accessed: 18/03/2007].

SwarmUsers (2007), 'Swarm development group', [on line]. Available from: http://www.swarm.org/wiki/Main_Page, [Last accessed: 18/03/2007].

Team, S. D. (2006), 'Simpy homepage', [online]. Available from: http://simpy.sourceforge.net/, [Last accessed: 18/03/2007].

The Green research unit, C. (2006), 'Cormas: Natural resources and agent-based simulations', [on line]. Available from: http://cormas.cirad.fr/indexeng.htm, [Last accessed: 18/03/2007].

Thompson, S. (2000), 'Zeus', [online]. Available from: http://labs.bt.com/projects/agents/zeus/, [Last accessed: 18/03/2007].

Tivnan, B. F. (2005), Coevolutionary dynamics and agent-based models in organizational science, *in* 'WSC '05: Proceedings of the 37th conference on Winter simulation', Executive Leadership Doctoral Program, Winter Simulation Conference, George Washington Univeristy, Ashburn, U.S.A, pp. 1013–1021.

Wallis, S. (n.d.), 'Sdml: a strictly declarative modelling language', [on line]. Available from: http://cfpm.org/sdml/, [Last accessed: 18/03/2007].

Welensky, U. (1999), 'Netlogo', [online]. Available from: http://ccl.northwestern.edu/netlogo/, [Last accessed: 18/03/2007].

Wooldridge, M. (2005), *MultiAgent Systems*, John Wiley & Sons Inc.

Wooldridge, M. & Jennings, N. R. (1995), 'Intelligent agents: Theory and practice', *Knowledge Engineering Review* **10**(2), 115–152.

Wright, S. (1932), The roles of mutation, inbreed, crossbreeding and selection in evolution, *in* D. F. Jones, ed., 'Proceeding of the Sixth International Congress on Genetics', pp. 356–366.